Präsibert

1.0

Generated by Doxygen 1.8.10

Sat Aug 15 2015 18:59:15

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 bb Namespace Reference

Namespaces

- EM2015

5.2 bb::EM2015 Namespace Reference

Classes

- class EMaudiorecorder
- class HDMI

5.3 Network Namespace Reference

Classes

- class ClientSocket

  ClientSocket class.
- class ConnectedClient

  Class for clients connected to the server.
- class ServerSocket

  ServerSocket class.

5.4 ServerAppl Namespace Reference

Classes

- class ByteStreamVerifier

  An object of this class can verify a QByteStream that has a hash at its and.
- class Listener

  Objects of this class represent a listener-client.
- class Logger
- class Master

Objects of this class represent a master-client.

- class MessageHandlerInterface
- class MessageRouter

  This class receives Message-objects and delivers them to target-objects.
- class Server

  An object of this class is the central component of the server-application.
- class UnspecifiedClient

  Objects of this class represent a client the has not shown if its a master-client or listener-client.

Chapter 6

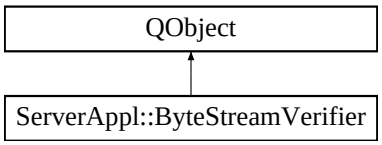
Class Documentation

6.1 ServerAppl::ByteStreamVerifier Class Reference

An object of this class can verify a QByteStream that has a hash at its and.

`#include "src/backend/ByteStreamVerifier.h"`

Inheritance diagram for ServerAppl::ByteStreamVerifier:



Public Slots

- void verifyCmdByteStream (QByteArray messageBytes, uint clientId)

  Takes a received command-byte-stream and tries to verify it with a MessageAuthenticator.
- void verifyDataByteStream (QByteArray messageBytes, uint clientId)

  Takes a received data-byte-stream and tries to verify it with a MessageAuthenticator.

Signals

- void cmdByteStreamVerified (QByteArray messageBytes, uint clientId)

  Will be emitted if a byte-stream from a command-port was successfully verified.
- void dataByteStreamVerified (QByteArray messageBytes, uint clientId)

  Will be emitted if a byte-stream from a data-port was successfully verified.
- void receivedInvalidByteStream (QByteArray bytes, uint clientId)

  Will be emitted if a byte-stream could not be verified.

Public Member Functions

- ByteStreamVerifier ()
- virtual ~ByteStreamVerifier ()
- bool addMessageAuthenticator (unsigned int clientId, MessageAuthenticator *authenticator)

  Adds a MessageAuthenticator-object for a specific client.
- bool removeMessageAuthenticator (unsigned int clientId)

  Removes a MessageAuthenticator-object for a specific client.

6.1.1 Detailed Description

An object of this class can verify a QByteStream that has a hash at its and.

This class implements the verification of received byte-streams from a specific client. Therefore it has a list with clientId's and corresponding MessageAuthenticator-objects. If a byte-stream from a client with an ID from that table was received the ByteStreamVerifier will try to check if the byte-stream is valid. If the byte-stream is not valid it will be deleted. Otherwise it will be forwarded. If no MessageAuthenticator for a clientId exist the byte-stream will always be forwarded.

Definition at line 32 of file ByteStreamVerifier.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 ServerAppl::ByteStreamVerifier::ByteStreamVerifier ( )

Definition at line 14 of file ByteStreamVerifier.cpp.

6.1.2.2 ServerAppl::ByteStreamVerifier::~ByteStreamVerifier ( ) [virtual]

Definition at line 19 of file ByteStreamVerifier.cpp.

6.1.3 Member Function Documentation

6.1.3.1 bool ServerAppl::ByteStreamVerifier::addMessageAuthenticator (unsigned int *clientId,* MessageAuthenticator * *authenticator*)

Adds a MessageAuthenticator-object for a specific client.

Definition at line 54 of file ByteStreamVerifier.cpp.

6.1.3.2 void ServerAppl::ByteStreamVerifier::cmdByteStreamVerified (QByteArray *messageBytes,* uint *clientId*) [signal]

Will be emitted if a byte-stream from a command-port was successfully verified.

6.1.3.3 void ServerAppl::ByteStreamVerifier::dataByteStreamVerified (QByteArray *messageBytes,* uint *clientId*) [signal]

Will be emitted if a byte-stream from a data-port was successfully verified.

6.1.3.4 void ServerAppl::ByteStreamVerifier::receivedInvalidByteStream (QByteArray *bytes,* uint *clientId*) [signal]

Will be emitted if a byte-stream could not be verified.

6.1.3.5 bool ServerAppl::ByteStreamVerifier::removeMessageAuthenticator (unsigned int *clientId*)

Removes a MessageAuthenticator-object for a specific client.

Returns

True if a MessageAuthenticator exists for the client and was successfully removed.

The object will NOT be deleted!

Definition at line 67 of file ByteStreamVerifier.cpp.

6.1.3.6 void ServerAppl::ByteStreamVerifier::verifyCmdByteStream (QByteArray *messageBytes*, uint *clientId*) [slot]

Takes a received command-byte-stream and tries to verify it with a MessageAuthenticator.

Definition at line 24 of file ByteStreamVerifier.cpp.

6.1.3.7 void ServerAppl::ByteStreamVerifier::verifyDataByteStream (QByteArray *messageBytes*, uint *clientId*) [slot]

Takes a received data-byte-stream and tries to verify it with a MessageAuthenticator.

Definition at line 40 of file ByteStreamVerifier.cpp.

The documentation for this class was generated from the following files:

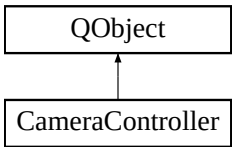
- ServerAppl/src/backend/ByteStreamVerifier.h
- ServerAppl/src/backend/ByteStreamVerifier.cpp

6.2 CameraController Class Reference

CameraController class.

`#include "Camera/CameraController.hpp"`

Inheritance diagram for CameraController:



Public Slots

- void start ()
  Activates the gesture control feature.
- void stop ()
  Deactivates the gesture control feature.
- void onGestureDetected (int value)
  Gets called when a gesture has been detected.
- void onError (QString e)
  Gets called when an error has occurred.

Signals

- void error (QString e)
  Emitted if an error has occured.
- void gestureDetected (int value)
  Emitted if a gesture is detected.

Public Member Functions

- CameraController (QObject *parent=NULL)
  Constructor for the CameraController class.
- virtual ~CameraController ()
  Deconstructor for the CameraController class.

6.2.1 Detailed Description

CameraController class.

The CameraController class is part of the gesture control component. It works as an interface for the rest of the application. When a CameraController object has been created, it can be used to activate and deactivate the gesture control feature.

Definition at line 24 of file CameraController.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 CameraController::CameraController (QObject * *parent* = NULL)

Constructor for the CameraController class.

Parameters

in	*parent*	Parent QObject that creates the CameraController object

This constructor creates a new QThread (but does not start it yet) and checks if the device has a front camera. If not an error signal is emitted.

Definition at line 11 of file CameraController.cpp.

6.2.2.2 CameraController::~CameraController ( ) [virtual]

Deconstructor for the CameraController class.

Definition at line 24 of file CameraController.cpp.

6.2.3 Member Function Documentation

6.2.3.1 void CameraController::error (QString *e*) [signal]

Emitted if an error has occured.

Parameters

out	*e*	QString with a description of the error

If something is not working (e. g. no front camera available), this signal is emitted. If the CameraController receives an error signal from its CameraProcessor object, this signal is used to transfer that error to the application.

6.2.3.2 void CameraController::gestureDetected (int *value*) [signal]

Emitted if a gesture is detected.

Parameters

out	*value*	Signals which gesture has been detected: -1 = to the left, +1 = to the right

If the CameraProcessor identifies a gesture, it emits a gestureDetected-Signal. This signal here is used to tranfer that signal to the application.

6.2.3.3 void CameraController::onError (QString *e*) [slot]

Gets called when an error has occurred.

Parameters

in	*e*	QString with a description of the error

If an error has occurred, this slot transfers the signal to the application

Definition at line 113 of file CameraController.cpp.

6.2.3.4 void CameraController::onGestureDetected (int *value*) [slot]

Gets called when a gesture has been detected.

Parameters

in	*value*	Indicates which gesture has been detected: -1 = to the left, +1 = to the right

If a gesture has been detected, this slot transfers the signal to the application

Definition at line 108 of file CameraController.cpp.

6.2.3.5 void CameraController::start ( ) [slot]

Activates the gesture control feature.

Creates a CameraProcessor object, moves it to a separate thread and calls its start()-function. This starts the scanning process for hand gestures.

Definition at line 47 of file CameraController.cpp.

6.2.3.6 void CameraController::stop ( ) [slot]

Deactivates the gesture control feature.

Deletes the CameraProcessor object and stops the thread. Thus stops scanning for hand gestures.

Definition at line 79 of file CameraController.cpp.

The documentation for this class was generated from the following files:

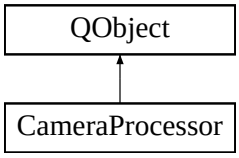
- Common/ClientServerShareLib/include/CameraController.hpp
- Common/ClientServerShareLib/src/Camera/CameraController.cpp

6.3 CameraProcessor Class Reference

CameraProcessor class.

`#include "Camera/CameraProcessor.hpp"`

Inheritance diagram for CameraProcessor:



Public Types

- enum Status { NOTHING, RIGHT, LEFT }

Public Slots

- void start ()

  Starts the camera and the gesture detection.
- void stop ()

  Stops the camera and the gesture detection.
- void process ()

  Analyzes the actual frame for a possible gesture.

Signals

- void gestureDetected (int value)

  Emitted if a gesture is detected.
- void error (QString e)

  Emitted if an error has occured.
- void imageReady ()

  Emitted if a new image is ready for processing.

Public Member Functions

- CameraProcessor (QObject *parent=NULL)

  Constructor for the CameraProcessor class.
- virtual ~CameraProcessor ()

  Deconstructor for the CameraController class.

6.3.1 Detailed Description

CameraProcessor class.

The CameraProcessor class is part of the gesture control component. When the gesture control feature gets activated, an instance of this class is created. It then enables the front camera unit and starts the viewfinder. Every frame gets analyzed and if a gesture is detected, an according signal is emitted. When the gesture control feature gets deactivated, this class stops scanning and frees the camera ressource.

Definition at line 27 of file CameraProcessor.hpp.

6.3.2 Member Enumeration Documentation

6.3.2.1 enum CameraProcessor::Status

Enumerator

NOTHING

RIGHT

LEFT

Definition at line 59 of file CameraProcessor.hpp.

6.3.3 Constructor & Destructor Documentation

6.3.3.1 CameraProcessor::CameraProcessor (QObject * *parent* = NULL)

Constructor for the CameraProcessor class.

Parameters

in	*parent*	Parent QObject that creates the CameraProcessor object

This constructor sets all member variables to default values and connects the imageReady-signal with the process()-slot, which is used for analyzing the frames.
To optimize the performance, try changing the following parameters:

- m_framerate: With a framerate of 16 (or less) frames per second every frame gets reliably processed, but maybe it could be set higher for better results.
- m_threshold: The threshold determines when a change in pixel value is deemed significant. The pixel values range from 255 (white) to 0 (black). Too small changes from one frame to the next may be caused by shadows etc., so they should be ignored.
- m_interval_percentage: Determines the percentage of a frame that counts as left or right (where gestures start and finish) and the interval in which a gesture is regarded continued from one frame to the next.

Definition at line 27 of file CameraProcessor.cpp.

6.3.3.2 CameraProcessor::~CameraProcessor ( ) [virtual]

Deconstructor for the CameraController class.

If the camera ressource is still in use, the stop()-procedure gets called.

Definition at line 47 of file CameraProcessor.cpp.

6.3.4 Member Function Documentation

6.3.4.1 void CameraProcessor::error (QString *e*) [signal]

Emitted if an error has occured.

Parameters

out	*e*	QString with a description of the error

If an error has occurred (e. g. it was not possible to set a specific resolution), this signal is emitted.

6.3.4.2 void CameraProcessor::gestureDetected (int *value*) [signal]

Emitted if a gesture is detected.

Parameters

out	*value*	Signals which gesture has been detected: -1 = to the left, +1 = to the right

If a gesture has been identified, this signal is emitted.

6.3.4.3 void CameraProcessor::imageReady ( ) [signal]

Emitted if a new image is ready for processing.

When a frame is available, the callback function viewfinder_callback() gets called. If the control variable m_busy is not set, this signal is emitted

6.3.4.4 void CameraProcessor::process ( ) [slot]

Analyzes the actual frame for a possible gesture.

When a frame is available, the callback function viewfinder_callback() gets called. If the control variable m_busy is not set, the signal imageReady() is emitted and process() gets called to analyze the new image which is stored in m_image.

Definition at line 381 of file CameraProcessor.cpp.

6.3.4.5 void CameraProcessor::start ( ) [slot]

Starts the camera and the gesture detection.

First enables the front camera unit by calling openCamera(), then adjusts the settings and starts the viewfinder by calling startVf().

Definition at line 56 of file CameraProcessor.cpp.

6.3.4.6 void CameraProcessor::stop ( ) [slot]

Stops the camera and the gesture detection.

First stops the viewfinder by calling stopVf(), then disables the front camera unit by calling closeCamera().

Definition at line 64 of file CameraProcessor.cpp.

The documentation for this class was generated from the following files:

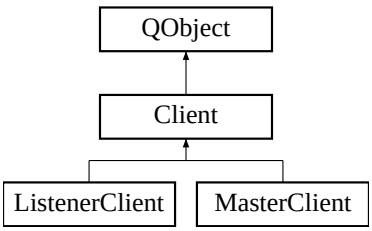
- Common/ClientServerShareLib/include/CameraProcessor.hpp
- Common/ClientServerShareLib/src/Camera/CameraProcessor.cpp

6.4 Client Class Reference

Basic implementation of Client.

`#include <Client.hpp>`

Inheritance diagram for Client:



Public Types

- enum LoginState {
  IDLE, CONNECTING, CONNECTED, TRYING,
  ACCEPTED, REJECTED }

  Loginstate of client.

Public Slots

- Message * setSlide (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_↩ types)

  Execute remote: Set slide to x.

- Message * parsePraesentation (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  Execute remote: Parse presentation from Message.
- Message * loginResponse (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  Execute remote: login response.
- Message * stopPraesentation (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  Execute remote: stop presentation.
- Q_INVOKABLE QString getLoginState ()

  Access loginstate as human readable string.
- Q_INVOKABLE QString getBasepath ()

  Access base path of presentation.
- Q_INVOKABLE void requestSlideChange (int offset)

  Request a slide change.
- Q_INVOKABLE void requestSlideChangeAbsolute (int slide)

  Request a slide change.
- Q_INVOKABLE void sendArbitraryCommand (QString cmd)

  Send arbitrary command to server. Implemented only for testpurposes.
- void connectionLost ()
- Q_INVOKABLE void deliverRecording (QString path)

  Initiate a delivery of an audio file via dataport of networklayer.
- Q_INVOKABLE void invokeRemote (Message *msg)
- Q_INVOKABLE void invokeRemote (Message *msg, bool cleanup)

  Remote procedure call.
- Q_INVOKABLE void login ()

  Initiate login to server.
- Q_INVOKABLE void logout ()

  Initiate logout from server.
- Q_INVOKABLE void connectToServer (QString addr, QString cmd_port, QString data_port)

  Connect to a server (on network layer) with command port and data port on specific IP address.
- void onNewSlideUrl (QUrl url)

  Slot for HDMI output if slide changed in presentation.

Signals

- void slideChanged (bb::cascades::Image img)

  Signal is thrown if slide changes and contains the image directly. Wrapper for QML access.
- void slideChangedUrl (QUrl url)

  Signal is thrown if slide changes and contains URL to new slide. Wrapper for QML access.
- void messageSent ()
- void loginStateChanged ()

  Signal is thrown if loginstate has changed.
- void wait (bool active)

  Signal is thrown if longer action is done and UI (or anything else) should indicate waiting state.
- void praesentationReady ()

  Signal is thrown if a valid presentation is loaded (either remotely (listener) or local (master). Wrapper for QML access.
- void noMoreSlides ()

  Signal is thrown if limits of slides is reached and next slide is accessed. Wrapper for QML access-.

Public Member Functions

- Client ()

  Constructor which initializes all members automatically and connects all signals and slots.
- virtual ~Client ()

  Destructor cleans up everything automatically.

Protected Attributes

- QMap< QString, remoteFunction > registerdFunctions

  Contains mapping from remote command to member function.
- bb::cascades::Image m_slide
- XMLMessageParser * xmlmp

  XML Message parser for commands.
- XMLMessageWriter * xmlmw

  XML Message writer to command port.
- XMLMessageParser * xmlmp_data

  XML Message parser for data.
- XMLMessageWriter * xmlmw_data

  XML Message writer to data port.
- Network::ClientSocket * cs

  Instance of networklayer.
- LoginState login_state

  Loginstate of the client.
- QString id

  ID of the client.
- Praesentation * prs

  Instance of presentation.s.
- bb::EM2015::HDMI * hdmi

  Instance of HDMI output wrapper..

6.4.1 Detailed Description

Basic implementation of Client.

Basic implementation of Client. Contains all members which are included in both Listener and Master.

Definition at line 40 of file Client.hpp.

6.4.2 Member Enumeration Documentation

6.4.2.1 enum Client::LoginState

Loginstate of client.

Enumerator

IDLE Client is not connected and not logged in

CONNECTING Client is connecting to server

CONNECTED Client is connected to server (only on TCP Layer

TRYING Client is trying to log in to server

ACCEPTED Client login was accepted

REJECTED Client login was rejected

Definition at line 45 of file Client.hpp.

6.4.3 Constructor & Destructor Documentation

6.4.3.1 Client::Client ( )

Constructor which initializes all members automatically and connects all signals and slots.

Definition at line 10 of file Client.cpp.

6.4.3.2 Client::~Client ( ) [virtual]

Destructor cleans up everything automatically.

Definition at line 55 of file Client.cpp.

6.4.4 Member Function Documentation

6.4.4.1 void Client::connectionLost ( ) [slot]

Definition at line 252 of file Client.cpp.

6.4.4.2 void Client::connectToServer (QString *addr,* QString *cmd_port,* QString *data_port*) [slot]

Connect to a server (on network layer) with command port and data port on specific IP address.

Connect to a server (on network layer) with command port and data port on specific IP address. Set loginstate to connecting.

Parameters

addr	IP address of server
cmd_port	Command port
data_port	Data port

Definition at line 228 of file Client.cpp.

6.4.4.3 void Client::deliverRecording (QString *path*) [slot]

Initiate a delivery of an audio file via dataport of networklayer.

Initiates a delivery of an audio file via the data port of the networklayer- Is primary called after the presentation is stopped or after a talk request is finished.

Parameters

path	Path to the file which should be delivered to the server.

Definition at line 286 of file Client.cpp.

6.4.4.4 QString Client::getBasepath ( ) [slot]

Access base path of presentation.

Access base path of presentation. Ment for GUI access (especially for audio recording in QML).

Returns

Basepath of presentation.

Definition at line 322 of file Client.cpp.

6.4.4.5 QString Client::getLoginState ( ) [slot]

Access loginstate as human readable string.

Access loginstate as human readable string. Ment for GUI access.

Returns

String which is congruent to current loginstate of Client.

Definition at line 196 of file Client.cpp.

6.4.4.6 void Client::invokeRemote (Message * *msg*) [slot]

Remote procedure call.

Parameters

msg	Message from Server.

See also

invokeRemote

Definition at line 70 of file Client.cpp.

6.4.4.7 void Client::invokeRemote (Message * *msg*, bool *cleanup*) [slot]

Remote procedure call.

Existential function in concept of remote procedure call. After Message is received by networklayer it is passed to the xml message parser. There it gets parsed to a message object which is then executed by this function. This function takes the message and looks up the corresponding member function which is ment to execute the remote command by looking it up in member registeredFunctions. Note that this is a whitelist approach; only commands which are registered can be executed, all others are silently dropped.

Parameters

msg	Message containing the command and payload.
cleanup	flag if the Message should be deleted after processing. Extension point for later versions where Message may be passed further afterwards.

Definition at line 76 of file Client.cpp.

6.4.4.8 void Client::login ( ) [slot]

Initiate login to server.

Definition at line 240 of file Client.cpp.

6.4.4.9 Message * Client::loginResponse (QMap< QString, QVariant > *parameters*, QMap< QString, QString > *parameter_types*) [slot]

Execute remote: login response.

Reacts to a response from server after login try. Message should contain a status of type string with value either "OK" or something else. Response is either OK or an error message which explains the reason for the error.

Parameters

parameters	Map of name and parameter value from Message
parameter_types	Map of name and type of parameter from Message

Returns

Response to receveived command.

Definition at line 164 of file Client.cpp.

6.4.4.10 void Client::loginStateChanged ( ) [signal]

Signal is thrown if loginstate has changed.

6.4.4.11 void Client::logout ( ) [slot]

Initiate logout from server.

Definition at line 309 of file Client.cpp.

6.4.4.12 void Client::messageSent ( ) [signal]

6.4.4.13 void Client::noMoreSlides ( ) [signal]

Signal is thrown if limits of slides is reached and next slide is accessed. Wrapper for QML access-.

6.4.4.14 void Client::onNewSlideUrl (QUrl *url*) [slot]

Slot for HDMI output if slide changed in presentation.

Slot for HDMI output if slide changed in presentation. Delivers URL to HDMI class because the HDMI class does not implement signal slot features.

Definition at line 315 of file Client.cpp.

6.4.4.15 Message * Client::parsePraesentation (QMap< QString, QVariant > *parameters*, QMap< QString, QString > *parameter_types*) [slot]

Execute remote: Parse presentation from Message.

Reacts to a parse presentation command. Message should contain following parameters: total_slides - int - number of total slides presentationID - string - id of presentation

Response is either OK or an error message which explains the reason for the error.

Parameters

parameters	Map of name and parameter value from Message
parameter_types	MAp of name and type of parameter from Message

Returns

Response to receveived command.

Definition at line 142 of file Client.cpp.

6.4.4.16 void Client::praesentationReady ( ) [signal]

Signal is thrown if a valid presentation is loaded (either remotely (listener) or local (master). Wrapper for QML access.

6.4.4.17 void Client::requestSlideChange (int *offset*) [slot]

Request a slide change.

Request a slide change from extern. For example, a UI Button or the gesture control can access those function either directly or via signal/slot.

Parameters

offset	relative position of the next slide.

See also

reqeustSlideChangeAbsolute

Definition at line 260 of file Client.cpp.

6.4.4.18 void Client::requestSlideChangeAbsolute (int *slide*) [slot]

Request a slide change.

Request a slide change from extern. Used for direct access to a specific slide.

Parameters

slide	absolute position of the next slide.

See also

requestSlideChange

Definition at line 269 of file Client.cpp.

6.4.4.19 void Client::sendArbitraryCommand (QString *cmd*) [slot]

Send arbitrary command to server. Implemented only for testpurposes.

Definition at line 278 of file Client.cpp.

6.4.4.20 Message * Client::setSlide (QMap< QString, QVariant > *parameters*, QMap< QString, QString > *parameter_types*) [slot]

Execute remote: Set slide to x.

Reacts to a slide change request. Message should contain slide parameter of type int which indicates number of slide to set the presentation to. Response is either OK or an error message which explains the reason for the error.

Parameters

parameters	Map of name and parameter value from Message

parameter_types	MAp of name and type of parameter from Message

Returns

Response to receveived command.

Definition at line 99 of file Client.cpp.

6.4.4.21 void Client::slideChanged (bb::cascades::Image *img*) [signal]

Signal is thrown if slide changes and contains the image directly. Wrapper for QML access.

6.4.4.22 void Client::slideChangedUrl (QUrl *url*) [signal]

Signal is thrown if slide changes and contains URL to new slide. Wrapper for QML access.

6.4.4.23 Message * Client::stopPraesentation (QMap< QString, QVariant > *parameters*, QMap< QString, QString > *parameter_types*) [slot]

Execute remote: stop presentation.

Reacts to a stop request. Repsonse is a ACK.

Parameters

parameters	Map of name and parameter value from Message
parameter_types	Map of name and type of parameter from Message

Returns

Response to receveived command.

Definition at line 153 of file Client.cpp.

6.4.4.24 void Client::wait (bool *active*) [signal]

Signal is thrown if longer action is done and UI (or anything else) should indicate waiting state.

6.4.5 Member Data Documentation

6.4.5.1 Network::ClientSocket* Client::cs [protected]

Instance of networklayer.

Definition at line 219 of file Client.hpp.

6.4.5.2 bb::EM2015::HDMI* Client::hdmi [protected]

Instance of HDMI output wrapper..

Definition at line 233 of file Client.hpp.

6.4.5.3 QString Client::id [protected]

ID of the client.

ID of client. If not logged in, it is set to "undefined_client" After login it is set to "master" for Master and id received from server for listener.

Definition at line 228 of file Client.hpp.

6.4.5.4 LoginState Client::login_state [protected]

Loginstate of the client.

Definition at line 221 of file Client.hpp.

6.4.5.5 bb::cascades::Image Client::m_slide [protected]

Definition at line 206 of file Client.hpp.

6.4.5.6 Praesentation* Client::prs [protected]

Instance of presentation.s.

Definition at line 230 of file Client.hpp.

6.4.5.7 QMap<QString, remoteFunction> Client::registerdFunctions [protected]

Contains mapping from remote command to member function.

Definition at line 204 of file Client.hpp.

6.4.5.8 XMLMessageParser* Client::xmlmp [protected]

XML Message parser for commands.

Definition at line 209 of file Client.hpp.

6.4.5.9 XMLMessageParser* Client::xmlmp_data [protected]

XML Message parser for data.

Definition at line 214 of file Client.hpp.

6.4.5.10 XMLMessageWriter* Client::xmlmw [protected]

XML Message writer to command port.

Definition at line 211 of file Client.hpp.

6.4.5.11 XMLMessageWriter* Client::xmlmw_data [protected]

XML Message writer to data port.

Definition at line 216 of file Client.hpp.

The documentation for this class was generated from the following files:

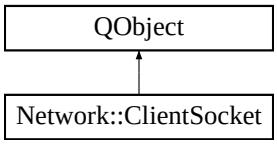
- Common/ClientServerShareLib/include/Client.hpp
- Common/ClientServerShareLib/src/Client/Client.cpp

6.5 Network::ClientSocket Class Reference

ClientSocket class.

`#include "Network/ClientSocket.h"`

Inheritance diagram for Network::ClientSocket:



Public Slots

- bool connectToServer (QString ipAddr_str, QString cmdPort_str, QString dataPort_str)

  Method for connecting the sockets to a server at the specified IP-Address.
- void disconnectFromServer ()

  Method to disconnect from server.
- int sendCmd (QByteArray data)

  Method that is used to send a command from the command socket.
- int sendData (QByteArray data)

  Method that is used to send data from the data socket.

Signals

- void connectedToCmdServer ()

  Signal that is emitted, when when the command socket successfully connected to its server.
- void connectedToDataServer ()

  Signal that is emitted, when when the data socket successfully connected to its server.
- void receivedCmd (QByteArray data)

  Signal that is emitted, when a new command is available at the command socket.
- void receivedData (QByteArray data)

  Signal that is emitted, when new data is available at the data socket.
- void lostConnection ()

  Signal that is emitted, when the connection was lost to one the server sockets.

Public Member Functions

- ClientSocket (QObject ∗)

  Constructor of the ClientSocket class.
- virtual ∼ClientSocket ()

  Destructor of the ClientSocket class.

6.5.1 Detailed Description

ClientSocket class.

Instantiates two TCP Client Sockets (command and data) that can connect to servers.
It uses 32 bit integers for determining the length of sent and received data.
The class provides several signals and slots for connection and data handling:

- signals:
  - connectedToCmdServer(): Emitted, when the command socket successfully connected to the server.
  - connectedToDataServer(): Emitted, when the data socket successfully connected to the server.
  - receivedCmd(): Emitted with ByteArray, when a new command is available.
  - receivedData(): Emitted with ByteArray, when new data is available.
  - lostConnection(): Emitted, when the connection to one of the servers (cmd or data) is lost.
- slots:
  - connectToServer(): Tries to establish a connection to a server with the IP and two ports (command and data port) that are given as parameter in QString format.
  - sendCmd(): Sends the command that is given as parameter to the command socket of the server.
  - sendData(): Sends the data that is given as parameter to the data socket of the server.
  - disconnectFromServer(): Disconnects the current connection to the server for both sockets.

Definition at line 46 of file ClientSocket.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 Network::ClientSocket::ClientSocket (QObject * *parent*)

Constructor of the ClientSocket class.

Initializes the command and data socket and connects signals and slots for connection and data handling.
Connects the signal *connected()* of the sockets with the handlers (*connectedToCmdServer()* and (*connectedTo↩DataServer()*) of this class.
Connects the signal *disconnected()* of the sockets with the slot *disconnectFromServer()* of this class.
Connects the signal *readyRead()* of the command socket with the slot *handleNewCmd()* of this class. Connects the signal *readyRead()* of the data socket with the slot *handleNewData()* of this class.

Definition at line 26 of file ClientSocket.cpp.

6.5.2.2 Network::ClientSocket::~ClientSocket ( ) [virtual]

Destructor of the ClientSocket class.

Closes the sockets and deletes them.

Definition at line 49 of file ClientSocket.cpp.

6.5.3 Member Function Documentation

6.5.3.1 void Network::ClientSocket::connectedToCmdServer ( ) [signal]

Signal that is emitted, when when the command socket successfully connected to its server.

6.5.3.2 void Network::ClientSocket::connectedToDataServer ( ) [signal]

Signal that is emitted, when when the data socket successfully connected to its server.

6.5.3.3 bool Network::ClientSocket::connectToServer (QString *ipAddr_str,* QString *cmdPort_str,* QString *dataPort_str*) [slot]

Method for connecting the sockets to a server at the specified IP-Address.

Parameters

in	*ipAddr_str*	IP-Address in QString format to connect to.
in	*cmdPort_str*	Specified port for the command connection in QString format.
in	*dataPort_str*	Specified port for the data connection in QString format.

Returns

Returns true, if the connection was established successfully. Otherwise returns false.

This method tries to connect to the server sockets at the IP-Address and the ports that were given as parameters. The method calls the disconnectFromServer() method, if the connection cannot be established.

Definition at line 69 of file ClientSocket.cpp.

6.5.3.4 void Network::ClientSocket::disconnectFromServer ( ) [slot]

Method to disconnect from server.

This method lets the socket disconnect from the server that it is connected to.
Emits the *lostConnection*-Signal after it disconnected from the server.

Definition at line 97 of file ClientSocket.cpp.

6.5.3.5 void Network::ClientSocket::lostConnection ( ) [signal]

Signal that is emitted, when the connection was lost to one the server sockets.

6.5.3.6 void Network::ClientSocket::receivedCmd (QByteArray *data*) [signal]

Signal that is emitted, when a new command is available at the command socket.

Parameters

out	*data*	Command that was read from the socket in QByteArray format.

6.5.3.7 void Network::ClientSocket::receivedData (QByteArray *data*) [signal]

Signal that is emitted, when new data is available at the data socket.

Parameters

out	*data*	Data that was read from the socket in QByteArray format.

6.5.3.8 int Network::ClientSocket::sendCmd (QByteArray *data*) [slot]

Method that is used to send a command from the command socket.

Parameters

in	*data*	Command that is send to the server.

Returns

Returns the number of bytes that were actually send to the server.

This method sends data from the command socket to the server.
A 32 bit integer with information about the data length is send first. Then the actual data follows.

Definition at line 119 of file ClientSocket.cpp.

6.5.3.9 int Network::ClientSocket::sendData (QByteArray *data*) [slot]

Method that is used to send data from the data socket.

Parameters

in	*data*	Data that is send to the server.

Returns

Returns the number of bytes that were actually send to the server.

This method sends data from the data socket to the server.
A 32 bit integer with information about the data length is send first. Then the actual data follows.

Definition at line 142 of file ClientSocket.cpp.

The documentation for this class was generated from the following files:

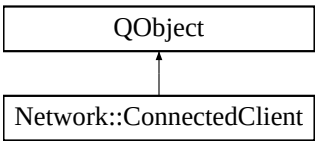
- Common/ClientServerShareLib/include/ClientSocket.h
- Common/ClientServerShareLib/src/Network/ClientSocket.cpp

6.6 Network::ConnectedClient Class Reference

Class for clients connected to the server.

`#include "Network/ConnectedClient.h"`

Inheritance diagram for Network::ConnectedClient:



Public Slots

- void process ()

  Method is called on start of the Thread.

Signals

- void newCmd (QByteArray data, uint clientID)

Signal that is emitted, whe an newcommanda is available fromthe commanda socket of the client.

- void newData (QByteArray data, uint clientID)

  Signal that is emitted, when new data is available from the data socket of the client.
- void disconnected (uint clientID)

  Signal that is emitted, when the client was disconnected.
- void finished ()

  Signal that is emitted, when the client is finished and ready for deletion.

Public Member Functions

- ConnectedClient (uint clientID)

  Constructor of the ConnectedClient class.
- virtual ~ConnectedClient ()

  Destructor of the ConnectedClient class.
- void setCmdSocket (QTcpSocket *tcpSocket)

  Method used to set the command socket.
- void setDataSocket (QTcpSocket *tcpSocket)

  Method used to set the data socket.
- bool hasCmdSocket ()

  Method that returns, whether the command socket is established already.
- bool hasDataSocket ()

  Method that returns, whether the data socket is established already.
- int sendCmd (QByteArray data)

  Sends a command to the connected client.
- int sendData (QByteArray data)

  Sends data to the connected client.
- void disconnectFromServer ()

  Closes the connection to the Server.
- uint getClientID ()

  Returns the clientID of the socket.
- QHostAddress getPeerAddress ()

  Returns the peer address of the connected client.

6.6.1 Detailed Description

Class for clients connected to the server.

Class for clients that connect to the Server Socket (ServerSocket class).
All of the objects that are created from this class are stored in an individual thread that is started when a new connection is established.
It uses 32 bit integers for determining the length of sent and received data.
The class provides several functions, signals and slots that are used to exchange data with a client:

- functions:
  - setCmdSocket(): Sets the socket that is given as parameter to the command socket.
  - setDataSocket(): Sets the socket that is given as parameter to the data socket.
  - hasCmdSocket(): Returns true, if the command socket is set up.
  - hasDataSocket(): Returns true, if the data socket is set up.
  - sendCmd(): Sends the ByteArray that is given as parameter to the clients command socket.
  - sendData(): Sends the ByteArray that is given as parameter to the clients data socket.

  - disconnectFromServer(): Disconnects both sockets from the servers sockets.
  - getClientID(): Returns the ID of the client.
  -
  - getPeerAddress(): Returns the peer address of the client.

- signals:
  - newCmd(): Emitted with the clientID and data when a new command is available from the clients command socket.
  - newData(): Emitted with the clientID and data when new data is available from the clients data socket.
  - disconnected(): Emitted with the clientID, when the connection to one of the clients sockets is lost.
  - finished(): Emitted, when the connection to the client is lost and the client socket can be destroyed.

- slots:
  - process(): This slot is connected to the start signal of the thread that stores the ConnectedClient object.

Definition at line 54 of file ConnectedClient.h.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 Network::ConnectedClient::ConnectedClient (uint *clientID*)

Constructor of the ConnectedClient class.

Parameters

in	*clientID*	ID of the client that is created.

Initializes the variables *m_clientID* with the value that was given as parameter.
Also initializes the booleans *hasCmdSocket* and *hasDataSocket* with the value *false* and *m_next_block_size_cmd* and *m_next_block_size_data* with the value *0*.
The constructor does not implement a parent object so that it can be moved into a QThread.

Definition at line 25 of file ConnectedClient.cpp.

6.6.2.2 Network::ConnectedClient::~ConnectedClient ( ) [virtual]

Destructor of the ConnectedClient class.

The destructor closes the sockets and deletes them.

Definition at line 39 of file ConnectedClient.cpp.

6.6.3 Member Function Documentation

6.6.3.1 void Network::ConnectedClient::disconnected (uint *clientID*) [signal]

Signal that is emitted, when the client was disconnected.

Parameters

out	*clientID*	ID of the client that lost the connection.

6.6.3.2 void Network::ConnectedClient::disconnectFromServer ( )

Closes the connection to the Server.

This class closes the connection to the servers sockets.

Definition at line 154 of file ConnectedClient.cpp.

6.6.3.3 void Network::ConnectedClient::finished ( ) [signal]

Signal that is emitted, when the client is finished and ready for deletion.

6.6.3.4 uint Network::ConnectedClient::getClientID ( )

Returns the clientID of the socket.

Returns

Returns the clientID of the socket.

Definition at line 166 of file ConnectedClient.cpp.

6.6.3.5 QHostAddress Network::ConnectedClient::getPeerAddress ( )

Returns the peer address of the connected client.

Returns

Returns the peer address of the connected client in QHostAddress format.

This function returns the peer address of the connected client.
The command socket is primarily used to determine the peer address.
If the command socket is not available, the data socket is used.
If none of the sockets is available, localhost is returned as peer address.

Definition at line 181 of file ConnectedClient.cpp.

6.6.3.6 bool Network::ConnectedClient::hasCmdSocket ( )

Method that returns, whether the command socket is established already.

Returns

Returns true, if the socket is set up and available.

Definition at line 88 of file ConnectedClient.cpp.

6.6.3.7 bool Network::ConnectedClient::hasDataSocket ( )

Method that returns, whether the data socket is established already.

Returns

Returns true, if the socket is set up and available.

Definition at line 98 of file ConnectedClient.cpp.

6.6.3.8 void Network::ConnectedClient::newCmd (QByteArray *data,* uint *clientID*) [signal]

Signal that is emitted, whe an newcommanda is available fromthe commanda socket of the client.

Parameters

out	*data*	Data in QByteArray format that is send out.
out	*clientID*	Own ID of the client that sends the data.

6.6.3.9 void Network::ConnectedClient::newData (QByteArray *data,* uint *clientID*) [signal]

Signal that is emitted, when new data is available from the data socket of the client.

Parameters

out	*data*	Data in QByteArray format that is send out.
out	*clientID*	Own ID of the client that sends the data.

6.6.3.10 void Network::ConnectedClient::process ( ) [slot]

Method is called on start of the Thread.

This method is called on start of the QThread that the ConnectedClient object was moved to.

Definition at line 206 of file ConnectedClient.cpp.

6.6.3.11 int Network::ConnectedClient::sendCmd (QByteArray *data*)

Sends a command to the connected client.

Parameters

in	*data*	Command in QByteArray format that is send.

Returns

Returns the amount of bytes that were actually sent to the client.

This method is used to send a command to the connected client that is given in QByteArray format as parameter.
A 32 bit integer with information about the data length is send first. Then the actual data follows.

Definition at line 113 of file ConnectedClient.cpp.

6.6.3.12 int Network::ConnectedClient::sendData (QByteArray *data*)

Sends data to the connected client.

Parameters

in	*data*	Data in QByteArray format that is send.

Returns

Returns the amount of bytes that were actually sent to the client.

This method is used to send data to the connected client that is given in QByteArray format as parameter.
A 32 bit integer with information about the data length is send first. Then the actual data follows.

Definition at line 136 of file ConnectedClient.cpp.

6.6.3.13 void Network::ConnectedClient::setCmdSocket (QTcpSocket * *tcpSocket*)

Method used to set the command socket.

Parameters

in	*tcpSocket*	New socket object that the clients command socket is assigned to.

Each time a new connection is established with the server, a new QTcpSocket object is created.
This object is then passed to this function as parameter.
Within this function, the new socket object is then assigned to the command socket.

Definition at line 56 of file ConnectedClient.cpp.

6.6.3.14 void Network::ConnectedClient::setDataSocket (QTcpSocket * *tcpSocket*)

Method used to set the data socket.

Parameters

in	*tcpSocket*	New socket object that the clients data socket is assigned to.

Each time a new connection is established with the server, a new QTcpSocket object is created.
This object is then passed to this function as parameter.
Within this function, the new socket object is then assigned to the data socket.

Definition at line 74 of file ConnectedClient.cpp.

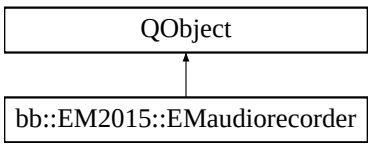
The documentation for this class was generated from the following files:

- Common/ClientServerShareLib/include/ConnectedClient.h
- Common/ClientServerShareLib/src/Network/ConnectedClient.cpp

6.7 bb::EM2015::EMaudiorecorder Class Reference

`#include <EMaudiorecorder.hpp>`

Inheritance diagram for bb::EM2015::EMaudiorecorder:



Public Member Functions

- EMaudiorecorder ()

  Initalizes the EMaudiorecorder v 1.0.
- ~EMaudiorecorder ()

  destructor
- char * record ()

  Start the recording.
- unsigned int stop ()

  Stop the recording.
- void LED_TEST ()

  LED test.

Public Attributes

- bool armed
- bool record_running
- int current_file

6.7.1 Detailed Description

Definition at line 34 of file EMaudiorecorder.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 bb::EM2015::EMaudiorecorder::EMaudiorecorder ( )

Initalizes the EMaudiorecorder v 1.0.

Initalizes the EMaudiorecorder: global variables and classes.

Definition at line 25 of file EMaudiorecorder.cpp.

6.7.2.2 bb::EM2015::EMaudiorecorder::~EMaudiorecorder ( )

destructor

Definition at line 48 of file EMaudiorecorder.cpp.

6.7.3 Member Function Documentation

6.7.3.1 void bb::EM2015::EMaudiorecorder::LED_TEST ( )

LED test.

This method toggles the green LED on. Unused in program, for testing purpose.

Definition at line 182 of file EMaudiorecorder.cpp.

6.7.3.2 char * bb::EM2015::EMaudiorecorder::record ( )

Start the recording.

This method implements the whole process of setting the file name, turning on the LED, preparing the recorder and starting it.

Returns

Adress of recording location as char array.

Definition at line 65 of file EMaudiorecorder.cpp.

6.7.3.3 unsigned int bb::EM2015::EMaudiorecorder::stop ( )

Stop the recording.

This method stops the recording and toggles the LED off.

Returns

Returns the duration of the recording for audio editing purpose.

Definition at line 161 of file EMaudiorecorder.cpp.

6.7.4 Member Data Documentation

6.7.4.1 bool bb::EM2015::EMaudiorecorder::armed

Definition at line 59 of file EMaudiorecorder.hpp.

6.7.4.2 int bb::EM2015::EMaudiorecorder::current_file

Definition at line 61 of file EMaudiorecorder.hpp.

6.7.4.3 bool bb::EM2015::EMaudiorecorder::record_running

Definition at line 60 of file EMaudiorecorder.hpp.

The documentation for this class was generated from the following files:

- Common/ClientServerShareLib/include/EMaudiorecorder.hpp
- Common/ClientServerShareLib/src/AudioRecorder/EMaudiorecorder.cpp

6.8 ExternalDisplay Class Reference

`#include <ExternalDisplay.hpp>`

Public Member Functions

- ExternalDisplay ()
- ~ExternalDisplay ()
- int open ()
- int close ()
- void setResolution (RESOLUTIONS_T res)
- RESOLUTIONS_T getResolution ()
- int showImage (bb::ImageData imageData)

6.8.1 Detailed Description

Definition at line 35 of file ExternalDisplay.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 ExternalDisplay::ExternalDisplay ( )

Definition at line 11 of file ExternalDisplay.cpp.

6.8.2.2 ExternalDisplay::~ExternalDisplay ( )

Definition at line 33 of file ExternalDisplay.cpp.

6.8.3 Member Function Documentation

6.8.3.1 int ExternalDisplay::close ( )

Definition at line 233 of file ExternalDisplay.cpp.

6.8.3.2 RESOLUTIONS_T ExternalDisplay::getResolution ( )

Definition at line 251 of file ExternalDisplay.cpp.

6.8.3.3 int ExternalDisplay::open ( )

Definition at line 38 of file ExternalDisplay.cpp.

6.8.3.4 void ExternalDisplay::setResolution (RESOLUTIONS_T *res*)

Definition at line 244 of file ExternalDisplay.cpp.

6.8.3.5 int ExternalDisplay::showImage (bb::ImageData *imageData*)

Definition at line 256 of file ExternalDisplay.cpp.

The documentation for this class was generated from the following files:

- Common/ClientServerShareLib/include/ExternalDisplay.hpp
- Common/ClientServerShareLib/src/HDMI/ExternalDisplay.cpp

6.9 bb::EM2015::HDMI Class Reference

`#include <HDMI.hpp>`

Public Member Functions

- HDMI (RESOLUTIONS_T hdmi_resolution)

  Initalizes the HDMI.
- virtual ~HDMI ()
- void show_slide (QUrl img_url)

  Show slide.
- void show_last_slide ()

  Show last slide.

6.9.1 Detailed Description

Definition at line 33 of file HDMI.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 bb::EM2015::HDMI::HDMI (RESOLUTIONS_T *hdmi_resolution*)

Initalizes the HDMI.

Initalizes the HDMI: global variables and classes.

- Screen resolution (RESOLUTIONS_T)

Definition at line 25 of file HDMI.cpp.

6.9.2.2 bb::EM2015::HDMI::~HDMI ( ) [virtual]

Definition at line 36 of file HDMI.cpp.

6.9.3 Member Function Documentation

6.9.3.1 void bb::EM2015::HDMI::show_last_slide ( )

Show last slide.

Show the last slide ("Presentation finished")

Returns

-none-

Definition at line 90 of file HDMI.cpp.

6.9.3.2 void bb::EM2015::HDMI::show_slide (QUrl *img_url*)

Show slide.

Update the output of HDMI, link to the picture in argument.

- QUrl img_url

  Returns

  -none-

Definition at line 51 of file HDMI.cpp.

The documentation for this class was generated from the following files:

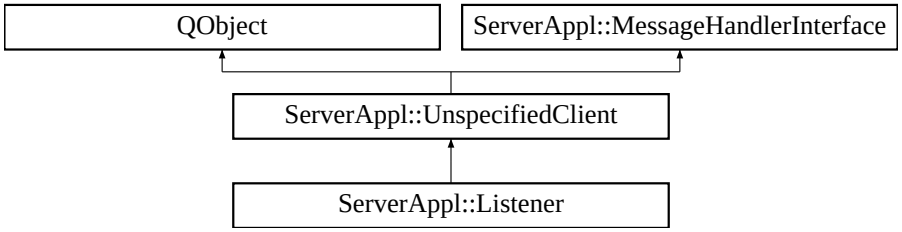
- Common/ClientServerShareLib/include/HDMI.hpp
- Common/ClientServerShareLib/src/HDMI/HDMI.cpp

6.10 ServerAppl::Listener Class Reference

Objects of this class represent a listener-client.

`#include "src/backend/Listener.h"`

Inheritance diagram for ServerAppl::Listener:



Signals

- void forwaredMessageToMaster (Message ∗msg, unsigned int clientId)
- void requestDeliverPresentation (unsigned int clientId)
- void writeAudioRecording (QString fileName, const QByteArray &recording)

Public Member Functions

- Listener (UnspecifiedClient ∗priorClientObject)
- Listener ()
- virtual ∼Listener ()
- void setHasPresentation (bool hasPresentation)

  Sets an indicator whether a presentation had been transmitted to the listener.
- bool getHasPresentation ()

  Indicates whether already a presentation has been transmitted to the listener.
- ClientType getClientType ()

  Returns the type of this client (for this class always ClientType_Lister).
- Message ∗ handleUnknownMessage (QString commandName, Message ∗msg)
- Message ∗ handleAcknowledge (QString commandName, Message ∗msg)
- Message ∗ handleReceivedAudio (QString commandName, Message ∗msg)

  Handles incomming audio-transmissions from the listener-client.

Static Public Member Functions

- static Listener ∗ createListener (UnspecifiedClient ∗client)

  Static function to generate a Listener-object basing on an UnspecifiedClient-object.

Additional Inherited Members

6.10.1 Detailed Description

Objects of this class represent a listener-client.

Objects of this class will represent a connected listener-client. It will be used mainly to hold some information about the listener (e.g. if already a presentation was delivered) and to receive audio-data from the listener.

Definition at line 28 of file Listener.h.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ServerAppl::Listener::Listener (UnspecifiedClient ∗ *priorClientObject*)

Definition at line 23 of file Listener.cpp.

6.10.2.2 ServerAppl::Listener::Listener ( )

Definition at line 19 of file Listener.cpp.

6.10.2.3 ServerAppl::Listener::~Listener ( ) [virtual]

Definition at line 36 of file Listener.cpp.

6.10.3 Member Function Documentation

6.10.3.1 Listener * ServerAppl::Listener::createListener (UnspecifiedClient * *client*) [static]

Static function to generate a Listener-object basing on an UnspecifiedClient-object.

Returns

A pointer to a new Listener-object will be returned (may be NULL!)

Definition at line 41 of file Listener.cpp.

6.10.3.2 void ServerAppl::Listener::forwaredMessageToMaster (Message * *msg*, unsigned int *clientId*) [signal]

6.10.3.3 ClientType ServerAppl::Listener::getClientType ( ) [virtual]

Returns the type of this client (for this class always ClientType_Lister).

Reimplemented from ServerAppl::UnspecifiedClient.

Definition at line 144 of file Listener.cpp.

6.10.3.4 bool ServerAppl::Listener::getHasPresentation ( )

Indicates whether already a presentation has been transmitted to the listener.

Returns

True if a presentation was transmitted.

Definition at line 139 of file Listener.cpp.

6.10.3.5 Message * ServerAppl::Listener::handleAcknowledge (QString *commandName*, Message * *msg*)

Definition at line 149 of file Listener.cpp.

6.10.3.6 Message * ServerAppl::Listener::handleReceivedAudio (QString *commandName*, Message * *msg*)

Handles incomming audio-transmissions from the listener-client.

The listener-shall transmit its audio-recordings over the data-port from a talk-request (Redeanfrage) to the server when the talk-request is finished. This message-handler emits the signal writeAudioRecording with the received audio-data as a QByteStream. The Server-object shall save the data afterwards to the file-system

Definition at line 106 of file Listener.cpp.

6.10.3.7 Message * ServerAppl::Listener::handleUnknownMessage (QString *commandName,* Message * *msg*) [virtual]

Implements ServerAppl::MessageHandlerInterface.

Definition at line 82 of file Listener.cpp.

6.10.3.8 void ServerAppl::Listener::requestDeliverPresentation (unsigned int *clientId*) [signal]

6.10.3.9 void ServerAppl::Listener::setHasPresentation (bool *hasPresentation*)

Sets an indicator whether a presentation had been transmitted to the listener.

Parameters

in	*hasPresentation*	True if a presentation was transmitted.

Definition at line 134 of file Listener.cpp.

6.10.3.10 void ServerAppl::Listener::writeAudioRecording (QString *fileName,* const QByteArray & *recording*) [signal]

The documentation for this class was generated from the following files:

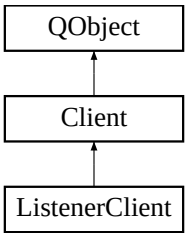
- ServerAppl/src/backend/Listener.h
- ServerAppl/src/backend/Listener.cpp

6.11 ListenerClient Class Reference

Implements extra functionality of ListenerClient.

`#include <ListenerClient.hpp>`

Inheritance diagram for ListenerClient:



Public Slots

- Message * redeanfrageResponse (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  Handle Response to talk-request.
- Message * redeanfrageFinish (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  Handle finishing of talk-request.
- Q_INVOKABLE void doRanf ()

  Initiates talk-request.
- Q_INVOKABLE void acceptRanf ()

  talk-request is relevant

- Q_INVOKABLE void rejectRanf ()
  talk request is not relevant anymore

Signals

- void ranfStateChanged (QString state)
  Rethrows signal of talk-request.
- void ranfAnswer ()

Public Member Functions

- ListenerClient ()
- virtual ~ListenerClient ()

Additional Inherited Members

6.11.1 Detailed Description

Implements extra functionality of ListenerClient.

Implements extra functionality of ListenerClient, mainly the possibility to do talk-requests.

Definition at line 20 of file ListenerClient.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 ListenerClient::ListenerClient ( )

Definition at line 10 of file ListenerClient.cpp.

6.11.2.2 ListenerClient::~ListenerClient ( ) [virtual]

Definition at line 25 of file ListenerClient.cpp.

6.11.3 Member Function Documentation

6.11.3.1 void ListenerClient::acceptRanf ( ) [slot]

talk-request is relevant

Definition at line 79 of file ListenerClient.cpp.

6.11.3.2 void ListenerClient::doRanf ( ) [slot]

Initiates talk-request.

Definition at line 71 of file ListenerClient.cpp.

6.11.3.3 void ListenerClient::ranfAnswer ( ) [signal]

6.11.3.4 void ListenerClient::ranfStateChanged (QString *state*) [signal]

Rethrows signal of talk-request.

6.11.3.5 Message * ListenerClient::redeanfrageFinish (QMap< QString, QVariant > *parameters*, QMap< QString, QString > *parameter_types*) [slot]

Handle finishing of talk-request.

Definition at line 61 of file ListenerClient.cpp.

6.11.3.6 Message * ListenerClient::redeanfrageResponse (QMap< QString, QVariant > *parameters*, QMap< QString, QString > *parameter_types*) [slot]

Handle Response to talk-request.

Definition at line 31 of file ListenerClient.cpp.

6.11.3.7 void ListenerClient::rejectRanf ( ) [slot]

talk request is not relevant anymore

Definition at line 88 of file ListenerClient.cpp.

The documentation for this class was generated from the following files:

- Client/ListenerClientAppl/src/ListenerClient/ListenerClient.hpp
- Client/ListenerClientAppl/src/ListenerClient/ListenerClient.cpp

6.12 ServerAppl::Logger Class Reference

`#include <Logger.h>`

Public Member Functions

- Logger ()
- virtual ~Logger ()
- void writeLogEntry (QString entry)
- void writeDebugLogEntry (const char file[], int line, QString entry)

6.12.1 Detailed Description

Definition at line 21 of file Logger.h.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 ServerAppl::Logger::Logger ( )

Definition at line 19 of file Logger.cpp.

6.12.2.2 ServerAppl::Logger::~Logger ( ) [virtual]

Definition at line 74 of file Logger.cpp.

6.12.3 Member Function Documentation

6.12.3.1 void ServerAppl::Logger::writeDebugLogEntry (const char *file[ ],* int *line,* QString *entry*)

6.12.3.2 void ServerAppl::Logger::writeLogEntry (QString *entry*)

Definition at line 87 of file Logger.cpp.

The documentation for this class was generated from the following files:

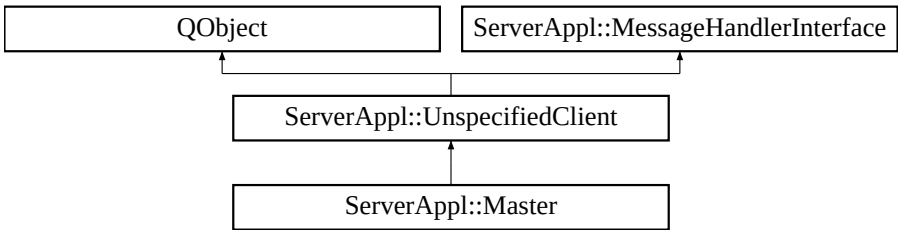
- ServerAppl/src/backend/Logger.h
- ServerAppl/src/backend/Logger.cpp

6.13 ServerAppl::Master Class Reference

Objects of this class represent a master-client.

`#include "src/backend/Master.h"`

Inheritance diagram for ServerAppl::Master:



Public Slots

- void onReceivedData (unsigned int receiverIdentifier)
- void onTransmitSlidesResponse (bool accepted)
- void authenticationTimeout ()

Signals

- void stopPresentation ()
- void forwardMessageToClient (Message ∗msg, unsigned int clientId)
- void receivedSlides ()
- void receivedSetSlide (int slideNumber)
- void authenticationFailed ()
- void authenticationSuccessfull ()
- void receivedPresentation (Praesentation ∗presentation, QMap< QString, QVariant > presentation↩ ParameterList, QMap< QString, QString > presentationParameterTypeList)
- void writeAudioRecording (QString fileName, const QByteArray &recording)

Public Member Functions

- Master ()
- Master (UnspecifiedClient ∗priorClientObject, QString nonce1)
- virtual ∼Master ()
- NONCE getNonce ()

Returns the complete nonce of the authentication-process.

- MessageAuthenticator * getMessageAuthenticator ()

  Returns a MessageAuthenticator-object which can be used to verify received messages from the master-client.
- ClientType getClientType ()

  Returns the type of this client (for this class always ClientType_Master).
- MasterAuthenticationState authenticationStm (MasterAuthenticationEvent event)

  This function contains a state-machine for the authentication-process.
- Message * handleUnknownMessage (QString commandName, Message *msg)
- Message * handleAuthenticationPhase3 (QString commandName, Message *msg)

  Handles phase-3-authentication-command. The received message-object (msg) will be deleted.
- Message * handleAuthenticationAcknowledge (QString commandName, Message *msg)

  Handles an authentication-acknowledge-command.
- Message * handleDataPresentation (QString commandName, Message *msg)

  This message-handler will save a received presentation.
- Message * handleSetSlide (QString commandName, Message *msg)

  This handles a set-slide-command of the master-client.
- Message * handleStopPresentation (QString commandName, Message *msg)

  Handles a stop-presentation-command.
- Message * handleReceivedAudio (QString commandName, Message *msg)

  Handles a received audio-transmission of the master-client.

Static Public Member Functions

- static Master * createMaster (UnspecifiedClient *client, QString nonce1)

  Static function to generate a Master-object basing on an UnspecifiedClient-object and a received nonce (part 1).
- static QString generateNonce (uint seed)

  A function to generate a nonce (e.g. nonce parte 2) basing on a seed.

Additional Inherited Members

6.13.1 Detailed Description

Objects of this class represent a master-client.

Objects of this class will represent a connected master-client. Some of the commands from the master will be handled by this class (e.g. authentication-commands, set-slide-commands or receiving data from the master-client).

Definition at line 56 of file Master.h.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 ServerAppl::Master::Master ( )

Definition at line 15 of file Master.cpp.

6.13.2.2 ServerAppl::Master::Master (UnspecifiedClient * *priorClientObject*, QString *nonce1*)

Definition at line 21 of file Master.cpp.

6.13.2.3 ServerAppl::Master::~Master ( ) [virtual]

Definition at line 49 of file Master.cpp.

6.13.3 Member Function Documentation

6.13.3.1 void ServerAppl::Master::authenticationFailed ( ) [signal]

6.13.3.2 MasterAuthenticationState ServerAppl::Master::authenticationStm (MasterAuthenticationEvent *event*)

This function contains a state-machine for the authentication-process.

Parameters

in	*event*	An event for the state-machine.

Returns

The new state of the state-machine.

This function can be used to ensure a correct authentication-process. Just give an event to the state-machine and check the returned state. If the returned state is the same as the expected new state everything is fine.

Definition at line 246 of file Master.cpp.

6.13.3.3 void ServerAppl::Master::authenticationSuccessfull ( ) [signal]

6.13.3.4 void ServerAppl::Master::authenticationTimeout ( ) [slot]

Definition at line 99 of file Master.cpp.

6.13.3.5 Master * ServerAppl::Master::createMaster (UnspecifiedClient * *client*, QString *nonce1*) [static]

Static function to generate a Master-object basing on an UnspecifiedClient-object and a received nonce (part 1).

Returns

A pointer to a new Master-object will be returned (may be NULL!)

Definition at line 56 of file Master.cpp.

6.13.3.6 void ServerAppl::Master::forwardMessageToClient (Message * *msg*, unsigned int *clientId*) [signal]

6.13.3.7 QString ServerAppl::Master::generateNonce (uint *seed*) [static]

A function to generate a nonce (e.g. nonce parte 2) basing on a seed.

Returns a string with a nonce.

Definition at line 61 of file Master.cpp.

6.13.3.8 ClientType ServerAppl::Master::getClientType ( ) [virtual]

Returns the type of this client (for this class always ClientType_Master).

Reimplemented from ServerAppl::UnspecifiedClient.

Definition at line 226 of file Master.cpp.

6.13.3.9 MessageAuthenticator * ServerAppl::Master::getMessageAuthenticator ( )

Returns a MessageAuthenticator-object which can be used to verify received messages from the master-client.

Definition at line 236 of file Master.cpp.

6.13.3.10 NONCE ServerAppl::Master::getNonce ( )

Returns the complete nonce of the authentication-process.

Returns

Returns the nonce which was transmitted by the master-client and the part that was generated on the server.

Definition at line 231 of file Master.cpp.

6.13.3.11 Message * ServerAppl::Master::handleAuthenticationAcknowledge (QString *commandName,* Message * *msg*)

Handles an authentication-acknowledge-command.

This command will be transmitted by the master after the server responded on the phase-3-command. The received message-object (msg) will be deleted.

Definition at line 126 of file Master.cpp.

6.13.3.12 Message * ServerAppl::Master::handleAuthenticationPhase3 (QString *commandName,* Message * *msg*)

Handles phase-3-authentication-command. The received message-object (msg) will be deleted.

Definition at line 137 of file Master.cpp.

6.13.3.13 Message * ServerAppl::Master::handleDataPresentation (QString *commandName,* Message * *msg*)

This message-handler will save a received presentation.

Usually this message-handler shall be registered for a data-port. If a presentation was received the server will be informed (signal receivedPresentation). The new presentation will be given to the Server-object. The Server-object is responsible for the distribution of the new presentation to the listener-clients. The received message-object (msg) will be deleted.

Definition at line 71 of file Master.cpp.

6.13.3.14 Message * ServerAppl::Master::handleReceivedAudio (QString *commandName,* Message * *msg*)

Handles a received audio-transmission of the master-client.

The audio-file will be given to the Server-object with the signal writeAudioRecording. The received message-object (msg) will be deleted.

Definition at line 180 of file Master.cpp.

6.13.3.15 Message * ServerAppl::Master::handleSetSlide (QString *commandName,* Message * *msg*)

This handles a set-slide-command of the master-client.

The server-object will be informed by the signal receivedSetSlide. The server-object is responsible for the forwarding of this command to the clients. The received message-object (msg) will be deleted.

Definition at line 86 of file Master.cpp.

6.13.3.16 Message * ServerAppl::Master::handleStopPresentation (QString *commandName,* Message * *msg*)

Handles a stop-presentation-command.

The Server-object will be informed of this command. The master-object will take no further actions. The received message-object (msg) will be deleted.

Definition at line 168 of file Master.cpp.

6.13.3.17 Message * ServerAppl::Master::handleUnknownMessage (QString *commandName,* Message * *msg*) [virtual]

Implements ServerAppl::MessageHandlerInterface.

Definition at line 106 of file Master.cpp.

6.13.3.18 void ServerAppl::Master::onReceivedData (unsigned int *receiverIdentifier*) [slot]

Definition at line 208 of file Master.cpp.

6.13.3.19 void ServerAppl::Master::onTransmitSlidesResponse (bool *accepted*) [slot]

Definition at line 241 of file Master.cpp.

6.13.3.20 void ServerAppl::Master::receivedPresentation (Praesentation * *presentation,* QMap< QString, QVariant > *presentationParameterList,* QMap< QString, QString > *presentationParameterTypeList*) [signal]

6.13.3.21 void ServerAppl::Master::receivedSetSlide (int *slideNumber*) [signal]

6.13.3.22 void ServerAppl::Master::receivedSlides ( ) [signal]

6.13.3.23 void ServerAppl::Master::stopPresentation ( ) [signal]

6.13.3.24 void ServerAppl::Master::writeAudioRecording (QString *fileName,* const QByteArray & *recording*) [signal]

The documentation for this class was generated from the following files:

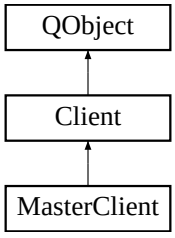
- ServerAppl/src/backend/Master.h
- ServerAppl/src/backend/Master.cpp

6.14 MasterClient Class Reference

Implements extra functionality of MasterClient.

`#include <MasterClient.hpp>`

Inheritance diagram for MasterClient:



Public Slots

- Message * loginResponse (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  react on authentication messages
- Message * redeanfrage (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  handle talk-request
- Message * redeanfrageAutoReject (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  auto-reject talk-request
- Message * redeanfrageFinal (QMap< QString, QVariant > parameters, QMap< QString, QString > parameter_types)

  handle if talk-request was relevant or not
- Q_INVOKABLE void authenticate ()

  initiate authentication
- void connectionLostMaster ()
- Q_INVOKABLE void clearRanf ()

  clear talk-requests
- Q_INVOKABLE void muteRanf ()

  mute talk-requests
- Q_INVOKABLE void acceptRanf ()

  accept talk-request
- Q_INVOKABLE void finishRanf ()

  finish talk-request
- Q_INVOKABLE void setKey (QString key)

  set the session key
- Q_INVOKABLE void selectPraesentation (QString path)

  read selected presentation from path
- Q_INVOKABLE void deliverPraesentation ()

  deliver presentation to server
- Q_INVOKABLE void requestStopPraesentation ()

  request a stop of the presentation
- Q_INVOKABLE void activateGesture (bool active)

  activate geasture controll

Signals

- void ranfMuteChanged (bool mute)

  thrown if talk-reqeust state of current request changed
- void ranfSizeChanged (int size)

  thrown if queue size changed
- void ranfFinalAnswer (QString answ)

  thrown if relevance information arrived
- void praesentationRunning (bool active)

  thrown to idicate presentation state

Public Member Functions

- MasterClient ()
- virtual ~MasterClient ()

Additional Inherited Members

6.14.1 Detailed Description

Implements extra functionality of MasterClient.

Implements extra functionality of MasterClient, mainly the possibility to queue and manage talk-requests, authentication and presentation controlling plus geasture control.

Definition at line 23 of file MasterClient.hpp.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 MasterClient::MasterClient ( )

Definition at line 10 of file MasterClient.cpp.

6.14.2.2 MasterClient::~MasterClient ( ) [virtual]

Definition at line 56 of file MasterClient.cpp.

6.14.3 Member Function Documentation

6.14.3.1 void MasterClient::acceptRanf ( ) [slot]

accept talk-request

Definition at line 270 of file MasterClient.cpp.

6.14.3.2 void MasterClient::activateGesture (bool *active*) [slot]

activate geasture controll

Definition at line 365 of file MasterClient.cpp.

6.14.3.3 void MasterClient::authenticate ( ) [slot]

initiate authentication

Definition at line 209 of file MasterClient.cpp.

6.14.3.4 void MasterClient::clearRanf ( ) [slot]

clear talk-requests

Definition at line 242 of file MasterClient.cpp.

6.14.3.5 void MasterClient::connectionLostMaster ( ) [slot]

Definition at line 222 of file MasterClient.cpp.

6.14.3.6 void MasterClient::deliverPraesentation ( ) [slot]

deliver presentation to server

Definition at line 351 of file MasterClient.cpp.

6.14.3.7 void MasterClient::finishRanf ( ) [slot]

finish talk-request

Definition at line 288 of file MasterClient.cpp.

6.14.3.8 Message * MasterClient::loginResponse (QMap< QString, QVariant > *parameters,* QMap< QString, QString > *parameter_types*) [slot]

react on authentication messages

Definition at line 63 of file MasterClient.cpp.

6.14.3.9 void MasterClient::muteRanf ( ) [slot]

mute talk-requests

Definition at line 257 of file MasterClient.cpp.

6.14.3.10 void MasterClient::praesentationRunning (bool *active*) [signal]

thrown to idicate presentation state

6.14.3.11 void MasterClient::ranfFinalAnswer (QString *answ*) [signal]

thrown if relevance information arrived

6.14.3.12 void MasterClient::ranfMuteChanged (bool *mute*) [signal]

thrown if talk-reqeust state of current request changed

6.14.3.13 void MasterClient::ranfSizeChanged (int *size*) [signal]

thrown if queue size changed

6.14.3.14 Message * MasterClient::redeanfrage (QMap< QString, QVariant > *parameters,* QMap< QString, QString > *parameter_types*) [slot]

handle talk-request

Definition at line 138 of file MasterClient.cpp.

6.14.3.15 Message * MasterClient::redeanfrageAutoReject (QMap< QString, QVariant > *parameters,* QMap< QString, QString > *parameter_types*) [slot]

auto-reject talk-request

Definition at line 160 of file MasterClient.cpp.

6.14.3.16 Message * MasterClient::redeanfrageFinal (QMap< QString, QVariant > *parameters*, QMap< QString, QString > *parameter_types*) [slot]

handle if talk-request was relevant or not

Definition at line 179 of file MasterClient.cpp.

6.14.3.17 void MasterClient::requestStopPraesentation ( ) [slot]

request a stop of the presentation

Definition at line 358 of file MasterClient.cpp.

6.14.3.18 void MasterClient::selectPraesentation (QString *path*) [slot]

read selected presentation from path

Definition at line 310 of file MasterClient.cpp.

6.14.3.19 void MasterClient::setKey (QString *key*) [slot]

set the session key

Definition at line 301 of file MasterClient.cpp.

The documentation for this class was generated from the following files:

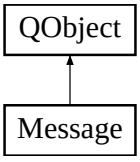
- Client/MasterClientAppl/src/MasterClient/MasterClient.hpp
- Client/MasterClientAppl/src/MasterClient/MasterClient.cpp

6.15 Message Class Reference

Implementation of a Message.

`#include <Message.hpp>`

Inheritance diagram for Message:



Public Member Functions

- Message ()
- Message (QString command, QString sender, QString receiver)
- virtual ~Message ()
- QString getCommand ()
- QString getSender ()
- QString getReceiver ()
- const QMap< QString, QVariant > * getParameters ()
- const QMap< QString, QString > * getParameterTypes ()
- void setParameterList (QMap< QString, QVariant > list)

- void setParameterTypeList (QMap< QString, QString > types)
- int addParameter (QString name, QString value)
- int addParameter (QString name, QDateTime value)
- int addParameter (QString name, int value)
- int addParameter (QString name, double value)
- int addParameter (QString name, QByteArray value)
- QDateTime getTimestamp ()
- void setTimestamp (QDateTime ts)

Friends

- class XMLMessageParser
- class XMLMessageWriter
- class Client
- class Praesentation

6.15.1 Detailed Description

Implementation of a Message.

Message: Contains Header:

- Sender
- Receiver
- Timestampt
- Command Contains Payload:
- Parameters
- Parameter types

Message is instantiated if needed and finally passed to a XML Writer which serializes the Message to XML and passes it to the network layer.^ Message is instantiated by XML Parser if network layer passed a valid XML Message to it.

A parameter can be added by calling addParameter with the name of the parameter and the payload. Message automatically recognizes the type of the parameter and adds the type to the parameter type map. For now following types are supported:

- integer
- decimal
- string
- date/time
- raw byte array (is converted to base64 automatically; has to be converted from base64 *manually*)

Member can either be accessed from friend classes or by getters and setter (payload is only accessible by friend-classes). If timestamp is requested and no timestamp is set yet the timestamp will be set to the time it is requested.

Definition at line 49 of file Message.hpp.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Message::Message ( )

Definition at line 19 of file Message.cpp.

6.15.2.2 Message::Message (QString *command,* QString *sender,* QString *receiver*)

Definition at line 10 of file Message.cpp.

6.15.2.3 Message::~Message ( ) [virtual]

Definition at line 24 of file Message.cpp.

6.15.3 Member Function Documentation

6.15.3.1 int Message::addParameter (QString *name,* QString *value*)

Definition at line 41 of file Message.cpp.

6.15.3.2 int Message::addParameter (QString *name,* QDateTime *value*)

Definition at line 57 of file Message.cpp.

6.15.3.3 int Message::addParameter (QString *name,* int *value*)

Definition at line 70 of file Message.cpp.

6.15.3.4 int Message::addParameter (QString *name,* double *value*)

Definition at line 83 of file Message.cpp.

6.15.3.5 int Message::addParameter (QString *name,* QByteArray *value*)

Definition at line 96 of file Message.cpp.

6.15.3.6 QString Message::getCommand ( )

Definition at line 136 of file Message.cpp.

6.15.3.7 const QMap< QString, QVariant > * Message::getParameters ( )

Definition at line 151 of file Message.cpp.

6.15.3.8 const QMap< QString, QString > * Message::getParameterTypes ( )

Definition at line 156 of file Message.cpp.

6.15.3.9 QString Message::getReceiver ( )

Definition at line 146 of file Message.cpp.

6.15.3.10 QString Message::getSender ( )

Definition at line 141 of file Message.cpp.

6.15.3.11 QDateTime Message::getTimestamp ( )

Definition at line 120 of file Message.cpp.

6.15.3.12 void Message::setParameterList (QMap< QString, QVariant > *list*)

Definition at line 29 of file Message.cpp.

6.15.3.13 void Message::setParameterTypeList (QMap< QString, QString > *types*)

Definition at line 35 of file Message.cpp.

6.15.3.14 void Message::setTimestamp (QDateTime *ts*)

Definition at line 131 of file Message.cpp.

6.15.4 Friends And Related Function Documentation

6.15.4.1 friend class Client [friend]

Definition at line 54 of file Message.hpp.

6.15.4.2 friend class Praesentation [friend]

Definition at line 55 of file Message.hpp.

6.15.4.3 friend class XMLMessageParser [friend]

Definition at line 52 of file Message.hpp.

6.15.4.4 friend class XMLMessageWriter [friend]

Definition at line 53 of file Message.hpp.

The documentation for this class was generated from the following files:

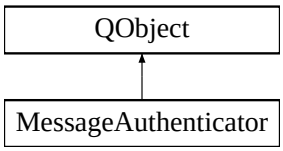
- Common/ClientServerShareLib/include/Message.hpp
- Common/ClientServerShareLib/src/Message/Message.cpp

6.16 MessageAuthenticator Class Reference

Provides a transparent interface to authenticate messages via HMAC given a key.

`#include <MessageAuthenticator.h>`

Inheritance diagram for MessageAuthenticator:



Public Slots

- void authenticateMessage (QByteArray msg)
- void setKey (QByteArray key)

Signals

- void messageAuthenticated (QByteArray msg)

Public Member Functions

- MessageAuthenticator ()
- virtual ~MessageAuthenticator ()
- QByteArray hmacSha1 (QByteArray baseString)
- QByteArray hmacSha1 (QByteArray key, QByteArray baseString)

6.16.1 Detailed Description

Provides a transparent interface to authenticate messages via HMAC given a key.

Provides a transparent interface to authenticate messages via HMAC given a key. Key has to be set before authentication. In addition the class provides HMAC algorithm. It accepts raw byte data (DATA) via slot and returns raw data concatenated with 28 byte base64 encoded mac (DATA||MAC) via signal. Example: For the Masterclient the Authenticator is placed between XML Writer and network layer (via reconnecting signals and slots) so every message automatically gets mac'ed.

Definition at line 22 of file MessageAuthenticator.h.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 MessageAuthenticator::MessageAuthenticator ( )

Definition at line 10 of file MessageAuthenticator.cpp.

6.16.2.2 MessageAuthenticator::~MessageAuthenticator ( ) [virtual]

Definition at line 15 of file MessageAuthenticator.cpp.

6.16.3 Member Function Documentation

6.16.3.1 void MessageAuthenticator::authenticateMessage (QByteArray *msg*) [slot]

Definition at line 58 of file MessageAuthenticator.cpp.

6.16.3.2 QByteArray MessageAuthenticator::hmacSha1 (QByteArray *baseString*)

Definition at line 20 of file MessageAuthenticator.cpp.

6.16.3.3 QByteArray MessageAuthenticator::hmacSha1 (QByteArray *key,* QByteArray *baseString*)

Definition at line 26 of file MessageAuthenticator.cpp.

6.16.3.4 void MessageAuthenticator::messageAuthenticated (QByteArray *msg*) [signal]

6.16.3.5 void MessageAuthenticator::setKey (QByteArray *key*) [slot]

Definition at line 67 of file MessageAuthenticator.cpp.

The documentation for this class was generated from the following files:

- Common/ClientServerShareLib/include/MessageAuthenticator.h
- Common/ClientServerShareLib/src/Message/Authentication/MessageAuthenticator.cpp

6.17 messageHandler Struct Reference

`#include <MessageRouter.h>`

Public Attributes

- ServerAppl::MessageHandlerInterface * object
- handleReceivedMessageFunction function

6.17.1 Detailed Description

Definition at line 20 of file MessageRouter.h.

6.17.2 Member Data Documentation

6.17.2.1 handleReceivedMessageFunction messageHandler::function

Definition at line 23 of file MessageRouter.h.

6.17.2.2 ServerAppl::MessageHandlerInterface* messageHandler::object

Definition at line 22 of file MessageRouter.h.

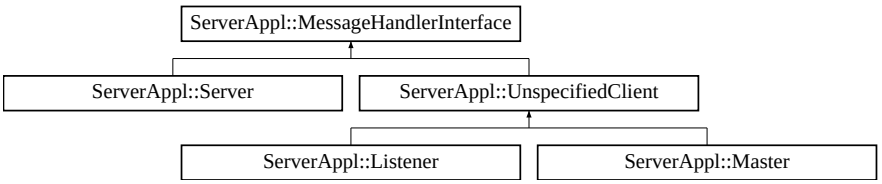
The documentation for this struct was generated from the following file:

- ServerAppl/src/backend/MessageRouter.h

6.18 ServerAppl::MessageHandlerInterface Class Reference

`#include <MessageHandlerInterface.h>`

Inheritance diagram for ServerAppl::MessageHandlerInterface:



Public Member Functions

- virtual Message * handleUnknownMessage (QString commandName, Message *msg)=0

6.18.1 Detailed Description

Definition at line 21 of file MessageHandlerInterface.h.

6.18.2 Member Function Documentation

6.18.2.1 virtual Message* ServerAppl::MessageHandlerInterface::handleUnknownMessage (QString *commandName,* Message * *msg*) [pure virtual]

Implemented in ServerAppl::Master, ServerAppl::Server, ServerAppl::UnspecifiedClient, and ServerAppl::Listener.

The documentation for this class was generated from the following file:

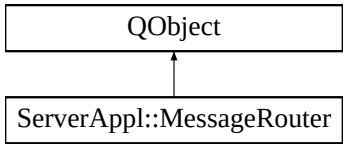
- ServerAppl/src/backend/MessageHandlerInterface.h

6.19 ServerAppl::MessageRouter Class Reference

This class receives Message-objects and delivers them to target-objects.

`#include "src/backend/MessageRouter.h"`

Inheritance diagram for ServerAppl::MessageRouter:



Public Slots

- void onMessageParsed (Message *message, uint clientId)

  This slot handles received Message-objects.

Signals

- void writeMessage (Message *message, uint clientId)

  This signal will be used to transmit response-messages for a received Message-object.

Public Member Functions

- MessageRouter ()
- ~MessageRouter ()
- bool registerMessageHandler (uint clientId, QString command, MessageHandlerInterface *object, handleReceivedMessageFunction function)

  Function to register a handler-function for a defined command and transmitter-client-id.
- bool registerMessageHandler (uint clientId, QString command, messageHandler handler)

  Function to register a handler-function for a defined command and transmitter-client-id.
- bool unregisterMessageHandler (uint clientId, QString command)

  Function to remove a message-handler from the registeredMessageHandlers-table.
- bool unregisterMessageHandlers (uint clientId)

  Function to remove all message-handler for commands from a client.
- bool addDirectRoute (QString receiver, uint receiverId)

  Function to register a direct route.
- bool removeDirectRoute (QString receiver)

  Function to remove a direct route.

6.19.1 Detailed Description

This class receives Message-objects and delivers them to target-objects.

Message-objects can be given to this class over the slot onMessageParsed. The MessageRouter will inspect the clientId of the transmitting client and will check in a table (registeredMessageHandlers) if handler-function for that Message-object exists. If a handler-function exist it will be called. If not the MessageRouter will try to find a direct route in the table directRoutingTable with the receiver-field of the Message-object. If no direkt-route was found this message-object will be deleted.

An object/class that want to receive Message-objects from this MessageRouter needs to emplement the class MessageHandlerInterface. If it does it can define handler-functions according to the type handleReceivedMessageFunction. These handler-functions can be added to the table registeredMessageHandlers with the function registerMessageHandler of the MessageRouter-class.

The signal writeMessage will be used to transmit response-messages.

Definition at line 44 of file MessageRouter.h.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 ServerAppl::MessageRouter::MessageRouter ( )

Definition at line 14 of file MessageRouter.cpp.

6.19.2.2 ServerAppl::MessageRouter::~MessageRouter ( )

Definition at line 20 of file MessageRouter.cpp.

6.19.3 Member Function Documentation

6.19.3.1 bool ServerAppl::MessageRouter::addDirectRoute (QString *receiver,* uint *receiverId*)

Function to register a direct route.

Parameters

in	*receiver*	This is a string with the receiver of a message.
in	*receiverId*	ClientId of the receiver.

Returns

Returns true if the direct route was registered successfully.

If for a received Message-object no message-handler could be found the MessageRouter will try to find the receiver-string of the Message-object in the table directRoutingTable. If the directRouting table contains the receiver-string the Message-object will be forwarded to the according clientId which is specified in the directRoutingTable.

Definition at line 108 of file MessageRouter.cpp.

6.19.3.2 void ServerAppl::MessageRouter::onMessageParsed (Message * *message*, uint *clientId*) [slot]

This slot handles received Message-objects.

Parameters

in	*message*	Pointer to the Message-object which was received (will be deleted whithin the function!)
in	*clientId*	ID of the client from which the Message-object was transmitted.

This slot will try to find a message-handler (basing on the clientId and the command-name of the Message-object). If no message-handler was found it will try to forward it on a direct-route (basing on the receiver-string of the Message-object and the directRoutingTable) to a specific client. If both failed (finding a message-handler and a direct-route) the Message-object will be deleted.

Definition at line 134 of file MessageRouter.cpp.

6.19.3.3 bool ServerAppl::MessageRouter::registerMessageHandler (uint *clientId*, QString *command*, MessageHandlerInterface * *object*, handleReceivedMessageFunction *function*)

Function to register a handler-function for a defined command and transmitter-client-id.

Parameters

in	*clientId*	ClientId of the client which transmitted the command to the server.
in	*command*	Name of the command which shall be handled.
in	*object*	Pointer to the object which contains the handler-function.
in	*function*	Pointer to the member-function which shall handle the command.

Returns

Returns true if the handler-function was registered successfully.

The function will try to add the given handler-function to the registeredMessageHandler-table. If already an entry for this command/client-combination exist a boolean false-value will be returned and no handler-function will be added.

Definition at line 32 of file MessageRouter.cpp.

6.19.3.4 bool ServerAppl::MessageRouter::registerMessageHandler (uint *clientId*, QString *command*, messageHandler *handler*)

Function to register a handler-function for a defined command and transmitter-client-id.

Parameters

in	*clientId*	ClientId of the client which transmitted the command to the server.
in	*command*	Name of the command which shall be handled.
in	*handler*	A structure with a pointer to the object with the handler-function and a function-pointer to the handler-function itself.

Returns

Returns true if the handler-function was registered successfully.

The function will try to add the given handler-function to the registeredMessageHandler-table. If already an entry for this command/client-combination exist a boolean false-value will be returned and no handler-function will be added.

Definition at line 43 of file MessageRouter.cpp.

6.19.3.5 bool ServerAppl::MessageRouter::removeDirectRoute (QString *receiver*)

Function to remove a direct route.

Parameters

in	*receiver*	A direct-route for this receiver will be removed.

Returns

Returns true if the route was found and successfully removed.

Definition at line 121 of file MessageRouter.cpp.

6.19.3.6 bool ServerAppl::MessageRouter::unregisterMessageHandler (uint *clientId,* QString *command*)

Function to remove a message-handler from the registeredMessageHandlers-table.

Parameters

in	*clientId*	ID of the client to which the message-handler belongs.
in	*command*	Command-name whose message-handler shall be removed.

Returns

Returns true if message-handler was found in the registeredMessageHandlers-table and successfully removed.

Definition at line 76 of file MessageRouter.cpp.

6.19.3.7 bool ServerAppl::MessageRouter::unregisterMessageHandlers (uint *clientId*)

Function to remove all message-handler for commands from a client.

Parameters

in	*clientId*	All message-handlers for this client will be removed.

Definition at line 95 of file MessageRouter.cpp.

6.19.3.8 void ServerAppl::MessageRouter::writeMessage (Message * *message,* uint *clientId*) [signal]

This signal will be used to transmit response-messages for a received Message-object.

Parameters

out	*message*	Pointer to the message-object which shall be transmitted.
out	*clientId*	ID of the client to which the Message-object shall be transmitted.

The documentation for this class was generated from the following files:

- ServerAppl/src/backend/MessageRouter.h
- ServerAppl/src/backend/MessageRouter.cpp

6.20 NONCE Struct Reference

`#include <Master.h>`

Public Attributes

- QString part_1
- QString part_2

6.20.1 Detailed Description

Definition at line 21 of file Master.h.

6.20.2 Member Data Documentation

6.20.2.1 QString NONCE::part_1

Definition at line 22 of file Master.h.

6.20.2.2 QString NONCE::part_2

Definition at line 23 of file Master.h.

The documentation for this struct was generated from the following file:

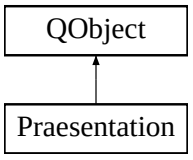
- ServerAppl/src/backend/Master.h

6.21 Praesentation Class Reference

Class to hold information about the presentation.

`#include <Praesentation.hpp>`

Inheritance diagram for Praesentation:



Public Slots

- void parsePraesentation (QMap< QString, QVariant > params, QMap< QString, QString > types)
  Parse presentation from parameters of a recevied message.
- Message * packPraesentation ()
  Pack presentation to a message.
- Message * packPraesentation (Message *msg)
  Pack and add presentation to an existing message.
- void appendSlide (QString path)
  Append slide to presentation.
- int getCurrentSlide ()
  Returns number of current slide (0 based).
- int getTotalSlides ()
  Returns total number of slides.
- void setSlide (int slide)
  Sets presentation to slide x.
- QString getPraesentationId ()
  Returns Presentation id.
- QString getBasepath ()
  Returns base path.
- void reset ()
  Resets all members.
- void stop ()
  Stops presentation and emits signal.

Signals

- void slideChanged (bb::cascades::Image)
  Signal with slide as image. Deprecated.
- void slideChangedUrl (QUrl url)
  Signal with slide as url.
- void praesentationParsed (Message *response)
  Signal if presentation is parsed.
- void parsing (bool active)
  Signal if parsing started/stopped.
- void praesentationReady ()
  Signal if presentation is ready to be started.
- void isRunning (bool active)
  Signal if presentation is running or not.

Public Member Functions

- Praesentation ()
- virtual ~Praesentation ()

6.21.1 Detailed Description

Class to hold information about the presentation.

This class hold the information about the current status of the presentation. It holds the URLs to the slides, the number of total slides and the current slide, as well as an identifier. It is implemented thread safe to enable access from multiple threads.

Definition at line 25 of file Praesentation.hpp.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 Praesentation::Praesentation ( )

Definition at line 10 of file Praesentation.cpp.

6.21.2.2 Praesentation::~Praesentation ( ) [virtual]

Definition at line 15 of file Praesentation.cpp.

6.21.3 Member Function Documentation

6.21.3.1 void Praesentation::appendSlide (QString *path*) [slot]

Append slide to presentation.

Definition at line 64 of file Praesentation.cpp.

6.21.3.2 QString Praesentation::getBasepath ( ) [slot]

Returns base path.

Definition at line 211 of file Praesentation.cpp.

6.21.3.3 int Praesentation::getCurrentSlide ( ) [slot]

Returns number of current slide (0 based).

Definition at line 52 of file Praesentation.cpp.

6.21.3.4 QString Praesentation::getPraesentationId ( ) [slot]

Returns Presentation id.

Definition at line 21 of file Praesentation.cpp.

6.21.3.5 int Praesentation::getTotalSlides ( ) [slot]

Returns total number of slides.

Definition at line 58 of file Praesentation.cpp.

6.21.3.6 void Praesentation::isRunning (bool *active*) [signal]

Signal if presentation is running or not.

6.21.3.7 Message * Praesentation::packPraesentation ( ) [slot]

Pack presentation to a message.

Definition at line 203 of file Praesentation.cpp.

6.21.3.8 Message * Praesentation::packPraesentation (Message * *msg*) [slot]

Pack and add presentation to an existing message.

Definition at line 179 of file Praesentation.cpp.

6.21.3.9 void Praesentation::parsePraesentation (QMap< QString, QVariant > *params*, QMap< QString, QString > *types*) [slot]

Parse presentation from parameters of a recevied message.

Definition at line 85 of file Praesentation.cpp.

6.21.3.10 void Praesentation::parsing (bool *active*) [signal]

Signal if parsing started/stopped.

6.21.3.11 void Praesentation::praesentationParsed (Message * *response*) [signal]

Signal if presentation is parsed.

6.21.3.12 void Praesentation::praesentationReady ( ) [signal]

Signal if presentation is ready to be started.

6.21.3.13 void Praesentation::reset ( ) [slot]

Resets all members.

Definition at line 27 of file Praesentation.cpp.

6.21.3.14 void Praesentation::setSlide (int *slide*) [slot]

Sets presentation to slide x.

Definition at line 151 of file Praesentation.cpp.

6.21.3.15 void Praesentation::slideChanged (bb::cascades::Image) [signal]

Signal with slide as image. Deprecated.

6.21.3.16 void Praesentation::slideChangedUrl (QUrl *url*) [signal]

Signal with slide as url.

6.21.3.17 void Praesentation::stop ( ) [slot]

Stops presentation and emits signal.

Definition at line 43 of file Praesentation.cpp.

The documentation for this class was generated from the following files:

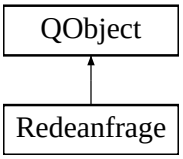
- Common/ClientServerShareLib/include/Praesentation.hpp
- Common/ClientServerShareLib/src/Praesentation/Praesentation.cpp

6.22 Redeanfrage Class Reference

Class containing one talk request and its state.

`#include <Redeanfrage.hpp>`

Inheritance diagram for Redeanfrage:



Public Types

- enum RedeanfrageState {
  PREPARATION, QUEUED, ACCEPTED, REJECTED,
  FINISHED }

Public Slots

- void prepare ()
  Prepare talk request.
- void queue (QString clientId)
  Queue talk request and set client id at once.
- void queue ()
  Queue talk request.
- void accept ()
  Accept talk request.
- void reject ()
  Reject talk request.
- void finish ()
  Finish talk request.
- Message * packRedeanfrage ()
  Pack talk request for sending.
- QString getClientId ()
  Returns client id.
- void setClientId (QString clientId)
  Set client id. necessary if client id is not known at creation time.

Signals

- void stateChanged (QString state)
  Signal if state has changed.

Public Member Functions

- Redeanfrage ()
- Redeanfrage (QString clientId)
- virtual ~Redeanfrage ()

6.22.1 Detailed Description

Class containing one talk request and its state.

Implementation of talk requests. Contains the state of the request as well as operations to change state or send the request. For detailed information see documentation's state diagram.

Definition at line 21 of file Redeanfrage.hpp.

6.22.2 Member Enumeration Documentation

6.22.2.1 enum Redeanfrage::RedeanfrageState

Enumerator

PREPARATION Preperation state

QUEUED queued and waiting for answer

ACCEPTED answer was an accept

REJECTED answer was a reject

FINISHED successfully finish talk request after accept

Definition at line 25 of file Redeanfrage.hpp.

6.22.3 Constructor & Destructor Documentation

6.22.3.1 Redeanfrage::Redeanfrage ( )

Definition at line 10 of file Redeanfrage.cpp.

6.22.3.2 Redeanfrage::Redeanfrage (QString *clientId*)

Definition at line 16 of file Redeanfrage.cpp.

6.22.3.3 Redeanfrage::~Redeanfrage ( ) [virtual]

Definition at line 22 of file Redeanfrage.cpp.

6.22.4 Member Function Documentation

6.22.4.1 void Redeanfrage::accept ( ) [slot]

Accept talk request.

Definition at line 53 of file Redeanfrage.cpp.

6.22.4.2 void Redeanfrage::finish ( ) [slot]

Finish talk request.

Definition at line 67 of file Redeanfrage.cpp.

6.22.4.3 QString Redeanfrage::getClientId ( ) [slot]

Returns client id.

Definition at line 85 of file Redeanfrage.cpp.

6.22.4.4 Message * Redeanfrage::packRedeanfrage ( ) [slot]

Pack talk request for sending.

Definition at line 74 of file Redeanfrage.cpp.

6.22.4.5 void Redeanfrage::prepare ( ) [slot]

Prepare talk request.

Definition at line 28 of file Redeanfrage.cpp.

6.22.4.6 void Redeanfrage::queue (QString *clientId*) [slot]

Queue talk request and set client id at once.

Definition at line 38 of file Redeanfrage.cpp.

6.22.4.7 void Redeanfrage::queue ( ) [slot]

Queue talk request.

Definition at line 46 of file Redeanfrage.cpp.

6.22.4.8 void Redeanfrage::reject ( ) [slot]

Reject talk request.

Definition at line 60 of file Redeanfrage.cpp.

6.22.4.9 void Redeanfrage::setClientId (QString *clientId*) [slot]

Set client id. necessary if client id is not known at creation time.

Definition at line 91 of file Redeanfrage.cpp.

6.22.4.10 void Redeanfrage::stateChanged (QString *state*) [signal]

Signal if state has changed.

The documentation for this class was generated from the following files:

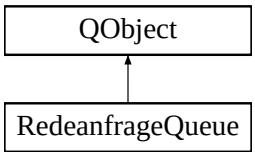
- Common/ClientServerShareLib/include/Redeanfrage.hpp
- Common/ClientServerShareLib/src/Redeanfrage/Redeanfrage.cpp

6.23 RedeanfrageQueue Class Reference

Thread safe implementation of a queue containing talk requests.

`#include <RedeanfrageQueue.hpp>`

Inheritance diagram for RedeanfrageQueue:



Public Slots

- int enqueue (Redeanfrage *ranf)

  Only enqueue element if its not already contained.
- Redeanfrage * dequeue ()
- void clear ()
- int getSize ()
- QString getClientIdAt (int i)

Signals

- void sizeChanged (int size)

Public Member Functions

- RedeanfrageQueue ()
- virtual ~RedeanfrageQueue ()

6.23.1 Detailed Description

Thread safe implementation of a queue containing talk requests.

Thread safe implementation of a queue containing talk requests. Distinct enqueueing per default.

Definition at line 19 of file RedeanfrageQueue.hpp.

6.23.2 Constructor & Destructor Documentation

6.23.2.1 RedeanfrageQueue::RedeanfrageQueue ( )

Definition at line 10 of file RedeanfrageQueue.cpp.

6.23.2.2 RedeanfrageQueue::~RedeanfrageQueue ( ) [virtual]

Definition at line 15 of file RedeanfrageQueue.cpp.

6.23.3 Member Function Documentation

6.23.3.1 void RedeanfrageQueue::clear ( ) [slot]

Definition at line 68 of file RedeanfrageQueue.cpp.

6.23.3.2 Redeanfrage * RedeanfrageQueue::dequeue ( ) [slot]

Definition at line 50 of file RedeanfrageQueue.cpp.

6.23.3.3 int RedeanfrageQueue::enqueue (Redeanfrage * *ranf*) [slot]

Only enqueue element if its not already contained.

Definition at line 22 of file RedeanfrageQueue.cpp.

6.23.3.4 QString RedeanfrageQueue::getClientIdAt (int *i*) [slot]

Definition at line 92 of file RedeanfrageQueue.cpp.

6.23.3.5 int RedeanfrageQueue::getSize ( ) [slot]

Definition at line 81 of file RedeanfrageQueue.cpp.

6.23.3.6 void RedeanfrageQueue::sizeChanged (int *size*) [signal]

The documentation for this class was generated from the following files:

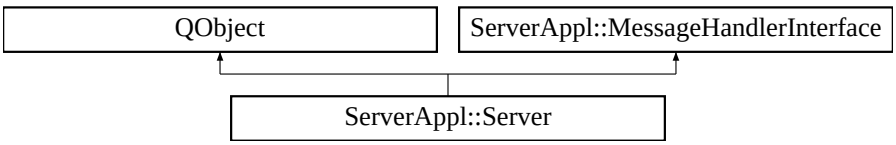
- Common/ClientServerShareLib/include/RedeanfrageQueue.hpp
- Common/ClientServerShareLib/src/Redeanfrage/RedeanfrageQueue.cpp

6.24 ServerAppl::Server Class Reference

An object of this class is the central component of the server-application.

`#include "src/backend/Server.h"`

Inheritance diagram for ServerAppl::Server:



Public Slots

- void onStopPresentation ()
- void onForwardMessageToClient (Message *msg, unsigned int clientId)
- void onForwaredMessageToMaster (Message *msg, unsigned int clientId)
- void onReceivedSetSlide (int slideNumber)
- void onNewClient (uint clientId)
- void onMasterAuthenticationFailed ()
- void onMasterAuthentificationSuccessfull ()
- void onReceivedPresentation (Praesentation *presentation, QMap< QString, QVariant > presentation↩ ParameterList, QMap< QString, QString > presentationParameterTypeList)
- void onDeliverPresentationToClient (unsigned int clientId)
- void onClientDisconnected (unsigned int clientId)
- void onNewIP (QString newIP)
- void onWriteAudioRecording (QString fileName, const QByteArray &recording)

Signals

- void sendCmdToID (Message *msg, uint clientID)
- void sendCmdMessageToMultClients (Message *data, QList< uint > clientIDs)
- void sendDataMessageToMultClients (Message *data, QList< uint > clientIDs)
- void sendCmdMessageToAll (Message *msg)
- void gotIpAddress (QString ipAddress)

Public Member Functions

- Server ()
- virtual ~Server ()
- QList< unsigned int > * getAllClientIdentifiers ()
- QList< unsigned int > * getListenerClientIdentifiers ()
- unsigned int getMasterClientIdentifier ()
- bool registerMaster (Master *master)
- bool unregisterMaster (Master *master)
- bool registerListener (Listener *listener)
- QString getIpAddress ()
- QString getCommandPort ()
- QString getDataPort ()
- Message * handleUnknownMessage (QString commandName, Message *msg)

Static Public Attributes

- static const char serverCommandPort [ ] = "1337"
- static const char serverDataPort [ ] = "1338"

6.24.1 Detailed Description

An object of this class is the central component of the server-application.

The Server-class contains all other components of the server-application (client-lists, ByteStreamVerifier, Server↩ Socket, MessageRouter, XmlMessageParser, XmlMessageWriter, ...).

On initialization all objects will be created and the network-module (ServerSocket) will be configured.

If the ServerSocket receives a new connection the slot onNewClient will be called. It will generate an Unspecified↩ Client-object and register all necessary message-handlers at the MessageRouter's (command- and data-router). Also the UnspecifiedClient-object will be inserted into the list connectedClients. When the UnspecifiedClient received a login-command (which shows the type of the client) the UnspecifiedClient-object will create a Master- or Listener-object and give it to the Server-object (registerMaster- or registerListener-function). Here it will be checked (e.g. for a Master-object if already exists one) and if everything is OK the object will be inserted into the listener↩ Clients-list or the masterClient-property. Afterwards the old UnspecifiedClient-object from the connectedClients-list will be replaced with the new one (Master- or Listener-object).

If the authentication-process of a master-object fails the slot onMasterAuthenticationFailed will delete all objects of the master-client and disconnect the network-connection.

If the master-object received a presentation the slot onReceivedPresentation will save the presentation in the Server-object an afterwards it will forward the presentation to all listenerClients which are successfully registered at the server. If further listener-Clients will connect the server the presentation will be transmitted to them right after successfull login.

If a Master- or Listener-object received an audio-recording the slot onWriteAudioRecording will write the audio-data into the file-system of the device. The filename will be given as a parameter of the slot. The path will be the same as the presentation used to save the slides.

Definition at line 61 of file Server.h.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 ServerAppl::Server::Server ( )

Definition at line 21 of file Server.cpp.

6.24.2.2 ServerAppl::Server::~Server ( ) [virtual]

Definition at line 155 of file Server.cpp.

6.24.3 Member Function Documentation

6.24.3.1 QList< unsigned int > * ServerAppl::Server::getAllClientIdentifiers ( )

Definition at line 411 of file Server.cpp.

6.24.3.2 QString ServerAppl::Server::getCommandPort ( )

Definition at line 440 of file Server.cpp.

6.24.3.3 QString ServerAppl::Server::getDataPort ( )

Definition at line 445 of file Server.cpp.

6.24.3.4 QString ServerAppl::Server::getIpAddress ( )

Definition at line 435 of file Server.cpp.

6.24.3.5 QList< unsigned int > * ServerAppl::Server::getListenerClientIdentifiers ( )

Definition at line 421 of file Server.cpp.

6.24.3.6 unsigned int ServerAppl::Server::getMasterClientIdentifier ( )

Definition at line 430 of file Server.cpp.

6.24.3.7 void ServerAppl::Server::gotIpAddress (QString *ipAddress*) [signal]

6.24.3.8 Message * ServerAppl::Server::handleUnknownMessage (QString *commandName,* Message * *msg*) [virtual]

Implements ServerAppl::MessageHandlerInterface.

Definition at line 177 of file Server.cpp.

6.24.3.9 void ServerAppl::Server::onClientDisconnected (unsigned int *clientId*) [slot]

Definition at line 297 of file Server.cpp.

6.24.3.10 void ServerAppl::Server::onDeliverPresentationToClient (unsigned int *clientId*) [slot]

Definition at line 369 of file Server.cpp.

6.24.3.11 void ServerAppl::Server::onForwardMessageToClient (Message * *msg,* unsigned int *clientId*) [slot]

Definition at line 182 of file Server.cpp.

6.24.3.12 void ServerAppl::Server::onForwaredMessageToMaster (Message * *msg,* unsigned int *clientId*) [slot]

Definition at line 190 of file Server.cpp.

6.24.3.13 void ServerAppl::Server::onMasterAuthenticationFailed ( ) [slot]

Definition at line 232 of file Server.cpp.

6.24.3.14 void ServerAppl::Server::onMasterAuthentificationSuccessfull ( ) [slot]

Definition at line 243 of file Server.cpp.

6.24.3.15 void ServerAppl::Server::onNewClient (uint *clientId*) [slot]

Definition at line 200 of file Server.cpp.

6.24.3.16 void ServerAppl::Server::onNewIP (QString *newIP*) [slot]

Definition at line 667 of file Server.cpp.

6.24.3.17 void ServerAppl::Server::onReceivedPresentation (Praesentation * *presentation,* QMap< QString, QVariant > *presentationParameterList,* QMap< QString, QString > *presentationParameterTypeList*) [slot]

Definition at line 257 of file Server.cpp.

6.24.3.18 void ServerAppl::Server::onReceivedSetSlide (int *slideNumber*) [slot]

Definition at line 268 of file Server.cpp.

6.24.3.19 void ServerAppl::Server::onStopPresentation ( ) [slot]

Definition at line 363 of file Server.cpp.

6.24.3.20 void ServerAppl::Server::onWriteAudioRecording (QString *fileName,* const QByteArray & *recording*) [slot]

Definition at line 388 of file Server.cpp.

6.24.3.21 bool ServerAppl::Server::registerListener (Listener * *listener*)

Definition at line 606 of file Server.cpp.

6.24.3.22 bool ServerAppl::Server::registerMaster (Master * *master*)

Definition at line 463 of file Server.cpp.

6.24.3.23 void ServerAppl::Server::sendCmdMessageToAll (Message * *msg*) [signal]

6.24.3.24 void ServerAppl::Server::sendCmdMessageToMultClients (Message * *data*, QList< uint > *clientIDs*) [signal]

6.24.3.25 void ServerAppl::Server::sendCmdToID (Message * *msg*, uint *clientID*) [signal]

6.24.3.26 void ServerAppl::Server::sendDataMessageToMultClients (Message * *data*, QList< uint > *clientIDs*) [signal]

6.24.3.27 bool ServerAppl::Server::unregisterMaster (Master * *master*)

Definition at line 563 of file Server.cpp.

6.24.4 Member Data Documentation

6.24.4.1 const char ServerAppl::Server::serverCommandPort = "1337" [static]

Definition at line 113 of file Server.h.

6.24.4.2 const char ServerAppl::Server::serverDataPort = "1338" [static]

Definition at line 114 of file Server.h.

The documentation for this class was generated from the following files:

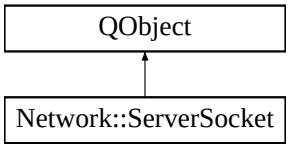
- ServerAppl/src/backend/Server.h
- ServerAppl/src/backend/Server.cpp

6.25 Network::ServerSocket Class Reference

ServerSocket class.

`#include "Network/ServerSocket.h"`

Inheritance diagram for Network::ServerSocket:



Public Slots

- bool beginListening (QString cmdPort_str, QString dataPort_str)

  Method that is called to start listening for incoming connections.
- void closeServer ()

  Signal to close the server.

- bool disconnectFromClient (uint clientID)

  Method to close the connection to a client.
- void sendCmdToAll (QByteArray data)

  Method for sending a command to all clients.
- void sendDataToAll (QByteArray data)

  Method for sending data to all clients.
- void sendCmdToMultClients (QByteArray data, QList< uint > clientIDs)

  Method for sending a command to a specified list of clients.
- void sendDataToMultClients (QByteArray data, QList< uint > clientIDs)

  Method for sending data to a specified list of clients.
- int sendCmdToID (QByteArray data, uint clientID)

  Method for sending a command to a client with specified ID.
- int sendDataToID (QByteArray data, uint clientID)

  Method for sending data to a client with specified ID.

Signals

- void newIP (QString newIP)

  Signal that is emitted, when the server is set up correctly and a correct IP was found.
- void clientDisconnect (uint clientID)

  Signal that is emitted, when the connection to a client was lost.
- void stoppedServer ()

  Signal that is emitted, when the server was stopped.
- void receivedCmdFromClient (QByteArray data, uint clientID)

  Signal that is emitted, when a new command was received from a client.
- void receivedDataFromClient (QByteArray data, uint clientID)

  Signal that is emitted, when new data was received from a client.
- void newClient (uint clientID)

  Signal that is emitted, when a new client connected to the server.

Public Member Functions

- ServerSocket (QObject *parent)

  Constructor of the ServerSocket class.
- virtual ~ServerSocket ()

  Destructor of the ServerSocket class.

6.25.1 Detailed Description

ServerSocket class.

Instantiates two TCP Server Sockets (command and data) that clients can connect to for communication.
The class provides several signals and slots for connection and data handling:

- signals:
  - newIP(): Emitted with current IP, when the server is set up.
  - clientDisconnect(): Emitted with clientID, when connection to a client is lost.
  - stoppedServer(): Emitted, when server is stopped.
  - receivedCmdFromClient(): Emitted with data and clientID, when a command was received from a client.
  - receivedDataFromClient(): Emitted with data and clientID, when data was received from a client.

- slots:
  - beginListening(): Calling this slot makes the server start to listen for incoming connections (command and data) of the ports that are given as parameter.
  - closeServer(): Shuts down the server.
  - sendCmdToAll(): Sends the command that is given as parameter to all of its clients command sockets.
  - sendDataToAll(): Sends the data that is given as parameter to all of its clients data sockets.
  - sendCmdToID(): Sends the command that is given as parameter to the client with the specified ID.
  - sendDataToID(): Sends the data that is given as parameter to the client with the specified ID.
  - sendCmdToMultClients(): Sends the command that is given as parameter to the IDs of clients that are given in a QList as parameter.
  - sendCmdToMultClients(): Sends the data that is given as parameter to the IDs of clients that are given in a QList as parameter.
  - disconnectFromClient(): Closes the connection to the client whose ID is given as parameter.

The ServerSocket manages all of the connected clients in a list (m_clientList) with a specific ID. For each client that establishes a connection to the server, an object of the *ConnectedClient*-class is created, pushed to an own thread and stored in m_clientList.
The *ConnectedClient*-class contains two QTcpSockets that the server can use to communicate with it's clients.

Definition at line 57 of file ServerSocket.h.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 Network::ServerSocket::ServerSocket (QObject * *parent*)

Constructor of the ServerSocket class.

Parameters

in	*parent*	Parent QObject that creates the server socket object.

The constructor initializes *m_clientID* (ID that is given to client) with its initial value *0*.
Also initializes both types of server sockets with *parent* as parameter.

Definition at line 27 of file ServerSocket.cpp.

6.25.2.2 Network::ServerSocket::~ServerSocket ( ) [virtual]

Destructor of the ServerSocket class.

Closes the server socket and deletes itself.

Definition at line 40 of file ServerSocket.cpp.

6.25.3 Member Function Documentation

6.25.3.1 bool Network::ServerSocket::beginListening (QString *cmdPort_str*, QString *dataPort_str*) [slot]

Method that is called to start listening for incoming connections.

Parameters

in	*cmdPort_str*	Listening port for incoming command connections in QString format.

in	*dataPort_str*	Listening port for incoming data connections in QString format.

Returns

Returns true, if the listening for incoming connections started successfully.

First the IP-Address of the server is located and the signal *newIP()* with the IP in QString format is emitted.
If no IP was found, localhost is used as IP-Address.
Afterwards initializes the TCP server sockets and starts listening for incoming connections on any address.
Also connects the signals *newConnection* of the server sockets with the handler slot *handleNewConnection()*.

Definition at line 59 of file ServerSocket.cpp.

6.25.3.2 void Network::ServerSocket::clientDisconnect (uint *clientID*) [signal]

Signal that is emitted, when the connection to a client was lost.

Parameters

out	*clientID*	ID of the client that the connection was lost to.

6.25.3.3 void Network::ServerSocket::closeServer ( ) [slot]

Signal to close the server.

Disconnects from all of the clients in *m_clientList* and closes both servers. Emits the signal *stoppedServer* afterwards.

Definition at line 139 of file ServerSocket.cpp.

6.25.3.4 bool Network::ServerSocket::disconnectFromClient (uint *clientID*) [slot]

Method to close the connection to a client.

Parameters

in	*clientID*	This method terminates the connection to a client that is connected to the server.

Definition at line 116 of file ServerSocket.cpp.

6.25.3.5 void Network::ServerSocket::newClient (uint *clientID*) [signal]

Signal that is emitted, when a new client connected to the server.

Parameters

out	*clientID*	ID of the client that connected to the server.

This Signal is only emitted, when **both** types of sockets (data and command) successfully connected to the server.

6.25.3.6 void Network::ServerSocket::newIP (QString *newIP*) [signal]

Signal that is emitted, when the server is set up correctly and a correct IP was found.

Parameters

out	*newIP*	IP-Address in QString format that was found.

6.25.3.7 void Network::ServerSocket::receivedCmdFromClient (QByteArray *data,* uint *clientID*) [signal]

Signal that is emitted, when a new command was received from a client.

Parameters

out	*data*	Command that was received from the client.
out	*clientID*	ID of the client that sent the command.

6.25.3.8 void Network::ServerSocket::receivedDataFromClient (QByteArray *data,* uint *clientID*) [signal]

Signal that is emitted, when new data was received from a client.

Parameters

out	*data*	Data that was received from the client.
out	*clientID*	ID of the client that sent the data.

6.25.3.9 void Network::ServerSocket::sendCmdToAll (QByteArray *data*) [slot]

Method for sending a command to all clients.

Parameters

in	*data*	Command that is send to the clients.

Sends a command to all clients.

Definition at line 161 of file ServerSocket.cpp.

6.25.3.10 int Network::ServerSocket::sendCmdToID (QByteArray *data,* uint *clientID*) [slot]

Method for sending a command to a client with specified ID.

Parameters

in	*data*	Command that is send to the clients.
in	*clientID*	ID of the client that the command is send to.

Sends a command to the client with the specified ID.

Definition at line 230 of file ServerSocket.cpp.

6.25.3.11 void Network::ServerSocket::sendCmdToMultClients (QByteArray *data,* QList< uint > *clientIDs*) [slot]

Method for sending a command to a specified list of clients.

Parameters

in	*data*	Command that is send to the clients.
in	*clientIDs*	QList with clientIDs to which the data is send.

Sends a command to all clients whose clientIDs are specified in the QList.

Definition at line 192 of file ServerSocket.cpp.

6.25.3.12 void Network::ServerSocket::sendDataToAll (QByteArray *data*) [slot]

Method for sending data to all clients.

Parameters

in	*data*	Data that is send to the clients.

Sends data to all clients.

Definition at line 176 of file ServerSocket.cpp.

6.25.3.13 int Network::ServerSocket::sendDataToID (QByteArray *data*, uint *clientID*) [slot]

Method for sending data to a client with specified ID.

Parameters

in	*data*	Data that is send to the clients.
in	*clientID*	ID of the client that the data is send to.

Sends data to the client with the specified ID.

Definition at line 251 of file ServerSocket.cpp.

6.25.3.14 void Network::ServerSocket::sendDataToMultClients (QByteArray *data*, QList< uint > *clientIDs*) [slot]

Method for sending data to a specified list of clients.

Parameters

in	*data*	Data that is send to the clients.
in	*clientIDs*	QList with clientIDs to which the data is send.

Sends data to all clients whose clientIDs are specified in the QList.

Definition at line 211 of file ServerSocket.cpp.

6.25.3.15 void Network::ServerSocket::stoppedServer ( ) [signal]

Signal that is emitted, when the server was stopped.

The documentation for this class was generated from the following files:

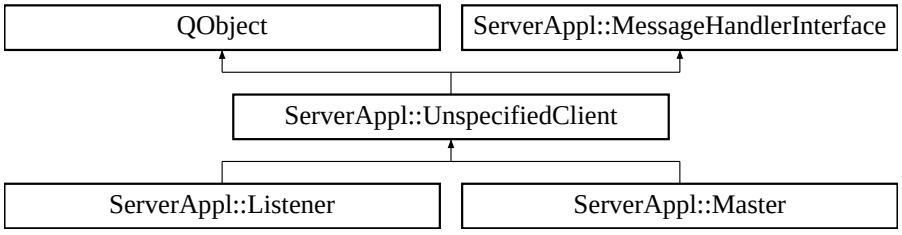
- Common/ClientServerShareLib/include/ServerSocket.h
- Common/ClientServerShareLib/src/Network/ServerSocket.cpp

6.26 ServerAppl::UnspecifiedClient Class Reference

Objects of this class represent a client the has not shown if its a master-client or listener-client.

`#include "src/backend/UnspecifiedClient.h"`

Inheritance diagram for ServerAppl::UnspecifiedClient:



Public Member Functions

- UnspecifiedClient ()
- UnspecifiedClient (Server *server, uint clientId, QString name)
- virtual ~UnspecifiedClient ()
- unsigned int getClientId ()

  Returns the unique ID of the client.
- virtual ClientType getClientType ()

  This function returns the type of the client.
- Message * handleUnknownMessage (QString commandName, Message *msg)
- Message * handleLoginMessages (QString commandName, Message *msg)

  Message-handler for login-commands of the client.
- Message * handleAuthPhase1 (QString commandName, Message *msg)

  Message-handler for auth-phase-1-commands.
- QString getName ()
- QTime getLastTimestamp ()
- Server * getServer ()

Protected Attributes

- Server * server
- uint clientId
- QString name
- QTime lastTimestamp

6.26.1 Detailed Description

Objects of this class represent a client the has not shown if its a master-client or listener-client.

The class UnspecifiedClient will represent a client for the time after the network-connection was established but no login-procedure was performed. In this time it is not clear what kind of client it is. After receiving a login-command it can be defined if its a master-client or a listener-client. It then will be transformed into an object of the classes Master or Listener. Both classes offer static functions which will build an Master/Listener-object basing on the data of a UnspecifiedClient-object.

Further this class is the base-class of Mater/Client-class. It defines a common interface for both classes and some basic-functionallity.

Definition at line 45 of file UnspecifiedClient.h.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 ServerAppl::UnspecifiedClient::UnspecifiedClient ( )

Definition at line 19 of file UnspecifiedClient.cpp.

6.26.2.2 ServerAppl::UnspecifiedClient::UnspecifiedClient (Server * *server,* uint *clientId,* QString *name*)

Definition at line 24 of file UnspecifiedClient.cpp.

6.26.2.3 ServerAppl::UnspecifiedClient::~UnspecifiedClient ( ) [virtual]

Definition at line 32 of file UnspecifiedClient.cpp.

6.26.3 Member Function Documentation

6.26.3.1 unsigned int ServerAppl::UnspecifiedClient::getClientId ( )

Returns the unique ID of the client.

Returns

Unsigned integer which is the ID of the client.

The ID's of the clients will be generated by the network-module.

Definition at line 146 of file UnspecifiedClient.cpp.

6.26.3.2 ClientType ServerAppl::UnspecifiedClient::getClientType ( ) [virtual]

This function returns the type of the client.

Returns

Will return ClientType_Unspecified, ClientType_Listener or ClientType_Master.

Reimplemented in ServerAppl::Master, and ServerAppl::Listener.

Definition at line 141 of file UnspecifiedClient.cpp.

6.26.3.3 QTime ServerAppl::UnspecifiedClient::getLastTimestamp ( )

Definition at line 156 of file UnspecifiedClient.cpp.

6.26.3.4 QString ServerAppl::UnspecifiedClient::getName ( )

Definition at line 151 of file UnspecifiedClient.cpp.

6.26.3.5 Server * ServerAppl::UnspecifiedClient::getServer ( )

Definition at line 161 of file UnspecifiedClient.cpp.

6.26.3.6 Message * ServerAppl::UnspecifiedClient::handleAuthPhase1 (QString *commandName,* Message * *msg*)

Message-handler for auth-phase-1-commands.

Only a master-client will transmit this command. It is the first command in the login-phase of the master-client. After receiving this command the UnspecifiedClient-object will be transformed into a Master-object and the Server-object will verify the new Master-object.

Definition at line 88 of file UnspecifiedClient.cpp.

6.26.3.7 Message * ServerAppl::UnspecifiedClient::handleLoginMessages (QString *commandName,* Message * *msg*)

Message-handler for login-commands of the client.

If the command transmitted a login-command to the server this function shall handle the command. Only listener-clients will transmit this command. After receiving this command the UnspecifiedClient-object will be transformed into a Listener-object. The new object will be verified by the Server-object.

Definition at line 48 of file UnspecifiedClient.cpp.

6.26.3.8 Message * ServerAppl::UnspecifiedClient::handleUnknownMessage (QString *commandName*, Message * *msg*) [virtual]

Implements ServerAppl::MessageHandlerInterface.

Definition at line 36 of file UnspecifiedClient.cpp.

6.26.4 Member Data Documentation

6.26.4.1 uint ServerAppl::UnspecifiedClient::clientId [protected]

Definition at line 98 of file UnspecifiedClient.h.

6.26.4.2 QTime ServerAppl::UnspecifiedClient::lastTimestamp [protected]

Definition at line 100 of file UnspecifiedClient.h.

6.26.4.3 QString ServerAppl::UnspecifiedClient::name [protected]

Definition at line 99 of file UnspecifiedClient.h.

6.26.4.4 Server* ServerAppl::UnspecifiedClient::server [protected]

Definition at line 97 of file UnspecifiedClient.h.

The documentation for this class was generated from the following files:

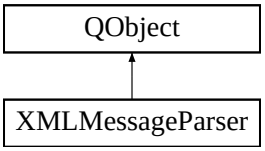
- ServerAppl/src/backend/UnspecifiedClient.h
- ServerAppl/src/backend/UnspecifiedClient.cpp

6.27 XMLMessageParser Class Reference

Parses Messages from XML to Message Object. Capable of separating between data an command/seperation by parent class. Throws signal if message was parsed.

`#include <XMLMessageParser.hpp>`

Inheritance diagram for XMLMessageParser:



Public Slots

- void parseMessage (QByteArray bytes)
- void parseCmdMessage (QByteArray bytes)
- void parseDataMessage (QByteArray bytes)
- void parseCmdMessage (QByteArray bytes, uint clientId)
- void parseDataMessage (QByteArray bytes, uint clientId)

Signals

- void messageParsed (Message ∗msg)
- void cmdMessageParsed (Message ∗msg)
- void dataMessageParsed (Message ∗msg)
- void cmdMessageParsed (Message ∗msg, uint clientId)
- void dataMessageParsed (Message ∗msg, uint clientId)

Public Member Functions

- XMLMessageParser ()
- virtual ∼XMLMessageParser ()

6.27.1 Detailed Description

Parses Messages from XML to Message Object. Capable of separating between data an command/seperation by parent class. Throws signal if message was parsed.

Definition at line 20 of file XMLMessageParser.hpp.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 XMLMessageParser::XMLMessageParser ( )

Definition at line 10 of file XMLMessageParser.cpp.

6.27.2.2 XMLMessageParser::∼XMLMessageParser ( ) [virtual]

Definition at line 15 of file XMLMessageParser.cpp.

6.27.3 Member Function Documentation

6.27.3.1 void XMLMessageParser::cmdMessageParsed (Message ∗ *msg*) [signal]

6.27.3.2 void XMLMessageParser::cmdMessageParsed (Message ∗ *msg*, uint *clientId*) [signal]

6.27.3.3 void XMLMessageParser::dataMessageParsed (Message ∗ *msg*) [signal]

6.27.3.4 void XMLMessageParser::dataMessageParsed (Message ∗ *msg*, uint *clientId*) [signal]

6.27.3.5 void XMLMessageParser::messageParsed (Message ∗ *msg*) [signal]

6.27.3.6 void XMLMessageParser::parseCmdMessage (QByteArray *bytes*) [slot]

Definition at line 25 of file XMLMessageParser.cpp.

6.27.3.7 void XMLMessageParser::parseCmdMessage (QByteArray *bytes*, uint *clientId*) [slot]

Definition at line 37 of file XMLMessageParser.cpp.

6.27.3.8 void XMLMessageParser::parseDataMessage (QByteArray *bytes*) [slot]

Definition at line 31 of file XMLMessageParser.cpp.

6.27.3.9 void XMLMessageParser::parseDataMessage (QByteArray *bytes*, uint *clientId*) [slot]

Definition at line 43 of file XMLMessageParser.cpp.

6.27.3.10 void XMLMessageParser::parseMessage (QByteArray *bytes*) [slot]

Definition at line 20 of file XMLMessageParser.cpp.

The documentation for this class was generated from the following files:

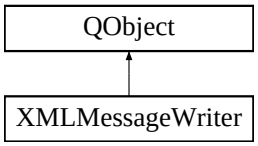
- Common/ClientServerShareLib/include/XMLMessageParser.hpp
- Common/ClientServerShareLib/src/Message/XML/XMLMessageParser.cpp

6.28 XMLMessageWriter Class Reference

Serializes Message object to XML. Capable of separating between data an command/seperation by parent class. Throws signal if message was written.

`#include <XMLMessageWriter.hpp>`

Inheritance diagram for XMLMessageWriter:



Public Slots

- void writeMessage (Message ∗msg)
- void writeCmdMessage (Message ∗msg)
- void writeDataMessage (Message ∗msg)
- void writeCmdMessage (Message ∗msg, uint clientId)
- void writeDataMessage (Message ∗msg, uint clientId)
- void writeCmdMessage (Message ∗msg, QList< uint > clientIDs)
- void writeDataMessage (Message ∗msg, QList< uint > clientIDs)

Signals

- void messageWritten (QByteArray msg)
- void cmdMessageWritten (QByteArray msg)
- void dataMessageWritten (QByteArray msg)
- void cmdMessageWritten (QByteArray msg, uint clientId)
- void dataMessageWritten (QByteArray msg, uint clientId)
- void cmdMessageWritten (QByteArray msg, QList< uint > clientIDs)
- void dataMessageWritten (QByteArray msg, QList< uint > clientIDs)

Public Member Functions

- XMLMessageWriter ()
- virtual ∼XMLMessageWriter ()

6.28.1 Detailed Description

Serializes Message object to XML. Capable of separating between data an command/seperation by parent class. Throws signal if message was written.

Definition at line 17 of file XMLMessageWriter.hpp.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 XMLMessageWriter::XMLMessageWriter ( )

Definition at line 10 of file XMLMessageWriter.cpp.

6.28.2.2 XMLMessageWriter::~XMLMessageWriter ( ) [virtual]

Definition at line 21 of file XMLMessageWriter.cpp.

6.28.3 Member Function Documentation

6.28.3.1 void XMLMessageWriter::cmdMessageWritten (QByteArray *msg*) [signal]

6.28.3.2 void XMLMessageWriter::cmdMessageWritten (QByteArray *msg*, uint *clientId*) [signal]

6.28.3.3 void XMLMessageWriter::cmdMessageWritten (QByteArray *msg*, QList< uint > *clientIDs*) [signal]

6.28.3.4 void XMLMessageWriter::dataMessageWritten (QByteArray *msg*) [signal]

6.28.3.5 void XMLMessageWriter::dataMessageWritten (QByteArray *msg*, uint *clientId*) [signal]

6.28.3.6 void XMLMessageWriter::dataMessageWritten (QByteArray *msg*, QList< uint > *clientIDs*) [signal]

6.28.3.7 void XMLMessageWriter::messageWritten (QByteArray *msg*) [signal]

6.28.3.8 void XMLMessageWriter::writeCmdMessage (Message * *msg*) [slot]

Definition at line 90 of file XMLMessageWriter.cpp.

6.28.3.9 void XMLMessageWriter::writeCmdMessage (Message * *msg*, uint *clientId*) [slot]

Definition at line 110 of file XMLMessageWriter.cpp.

6.28.3.10 void XMLMessageWriter::writeCmdMessage (Message * *msg*, QList< uint > *clientIDs*) [slot]

Definition at line 129 of file XMLMessageWriter.cpp.

6.28.3.11 void XMLMessageWriter::writeDataMessage (Message * *msg*) [slot]

Definition at line 100 of file XMLMessageWriter.cpp.

6.28.3.12 void XMLMessageWriter::writeDataMessage (Message * *msg*, uint *clientId*) [slot]

Definition at line 120 of file XMLMessageWriter.cpp.

6.28.3.13 void XMLMessageWriter::writeDataMessage (Message * *msg*, QList< uint > *clientIDs*) [slot]

Definition at line 139 of file XMLMessageWriter.cpp.

6.28.3.14 void XMLMessageWriter::writeMessage (Message * *msg*) [slot]

Definition at line 80 of file XMLMessageWriter.cpp.

The documentation for this class was generated from the following files:

- Common/ClientServerShareLib/include/XMLMessageWriter.hpp
- Common/ClientServerShareLib/src/Message/XML/XMLMessageWriter.cpp

Chapter 7

File Documentation

7.1 Client/ListenerClientAppl/src/ListenerClient/ListenerClient.cpp File Reference

```
#include "ListenerClient.hpp"
```

7.2 Client/ListenerClientAppl/src/ListenerClient/ListenerClient.hpp File Reference

```
#include <QObject>
#include "Client.hpp"
#include "commands.hpp"
#include "Redeanfrage.hpp"
```

Classes

- class ListenerClient

  Implements extra functionality of ListenerClient.

7.3 Client/MasterClientAppl/src/MasterClient/MasterClient.cpp File Reference

```
#include "MasterClient.hpp"
```

7.4 Client/MasterClientAppl/src/MasterClient/MasterClient.hpp File Reference

```
#include <QObject>
#include "Client.hpp"
#include "MessageAuthenticator.h"
#include "commands.hpp"
#include "Redeanfrage.hpp"
#include "RedeanfrageQueue.hpp"
#include "CameraController.hpp"
```

Classes

- class MasterClient

  Implements extra functionality of MasterClient.

7.5 Common/ClientServerShareLib/include/CameraController.hpp File Reference

```
#include <QObject>
#include <QThread>
#include "CameraProcessor.hpp"
```

Classes

- class CameraController

  CameraController class.

7.6 Common/ClientServerShareLib/include/CameraProcessor.hpp File Reference

```
#include <stdlib.h>
#include <QObject>
#include <camera/camera_api.h>
#include <QDebug>
#include "opencv2/core/core.hpp"
```

Classes

- class CameraProcessor

  CameraProcessor class.

7.7 Common/ClientServerShareLib/include/Client.hpp File Reference

```
#include <QObject>
#include <QtGui/QImage>
#include <QtCore>
#include <bb/ImageData>
#include <bb/cascades/Image>
#include <bb/utility/ImageConverter>
#include "Message.hpp"
#include "XMLMessageParser.hpp"
#include "XMLMessageWriter.hpp"
#include "ClientSocket.h"
#include "Praesentation.hpp"
#include "commands.hpp"
#include "EMaudiorecorder.hpp"
#include "HDMI.hpp"
```

Classes

- class Client

  Basic implementation of Client.

Typedefs

- typedef Message *(Client::* remoteFunction) (QMap< QString, QVariant > parameters, QMap< QString, QString > paramter_types)

7.7.1 Typedef Documentation

7.7.1.1 typedef Message*(Client::* remoteFunction) (QMap< QString, QVariant > parameters, QMap< QString, QString > paramter_types)

Type of function capable of being executed remotely

Definition at line 33 of file Client.hpp.

7.8 Common/ClientServerShareLib/include/clientservershare lib_global.hpp File Reference

```
#include <QtCore/qglobal.h>
```

Macros

- #define CLIENTSERVERSHARELIB_EXPORT Q_DECL_IMPORT

7.8.1 Macro Definition Documentation

7.8.1.1 #define CLIENTSERVERSHARELIB_EXPORT Q_DECL_IMPORT

Definition at line 26 of file clientservershare lib_global.hpp.

7.9 Common/ClientServerShareLib/include/ClientSocket.h File Reference

```
#include <QObject>
#include <QtNetwork/QTcpSocket>
#include <QString>
#include <QByteArray>
#include <QtCore>
```

Classes

- class Network::ClientSocket

  ClientSocket class.

Namespaces

- Network

7.10 Common/ClientServerShareLib/include/commands.hpp File Reference

Macros

- #define CMD_UNKNOWN "unknown_command"
- #define CMD_SET_SLIDE "set_slide"
- #define CMD_SET_PRAESENTATION "praesentation"
- #define CMD_STOP_PRAESENTATION "stop_praesentation"
- #define DATA_PRAESENTATION "deliver_praesentation"
- #define CMD_LOGIN "login"
- #define CMD_LOGIN_RESP "login_resp"
- #define CMD_AUTH_PHASE1 "auth_phase1"
- #define CMD_AUTH_PHASE2 "auth_phase2"
- #define CMD_AUTH_PHASE3 "auth_phase3"
- #define CMD_AUTH_PHASE4 "auth_phase4"
- #define CMD_ACK_RESPONSE "ack"
- #define CMD_RANF_ASK "redeanfrage_request"
- #define CMD_RANF_RESP "redeanfrage_antwort"
- #define CMD_RANF_RE_RESP "redeanfrage_finale_antwort"
- #define CMD_RANF_FINISH "redeanfrage_finish"
- #define DATA_AUDIO "deliver_audio"

7.10.1 Macro Definition Documentation

7.10.1.1 #define CMD_ACK_RESPONSE "ack"

Definition at line 32 of file commands.hpp.

7.10.1.2 #define CMD_AUTH_PHASE1 "auth_phase1"

Definition at line 28 of file commands.hpp.

7.10.1.3 #define CMD_AUTH_PHASE2 "auth_phase2"

Definition at line 29 of file commands.hpp.

7.10.1.4 #define CMD_AUTH_PHASE3 "auth_phase3"

Definition at line 30 of file commands.hpp.

7.10.1.5 #define CMD_AUTH_PHASE4 "auth_phase4"

Definition at line 31 of file commands.hpp.

7.10.1.6 #define CMD_LOGIN "login"

Definition at line 24 of file commands.hpp.

7.10.1.7 #define CMD_LOGIN_RESP "login_resp"

Definition at line 25 of file commands.hpp.

7.10.1.8 #define CMD_RANF_ASK "redeanfrage_request"

Definition at line 35 of file commands.hpp.

7.10.1.9 #define CMD_RANF_FINISH "redeanfrage_finish"

Definition at line 38 of file commands.hpp.

7.10.1.10 #define CMD_RANF_RE_RESP "redeanfrage_finale_antwort"

Definition at line 37 of file commands.hpp.

7.10.1.11 #define CMD_RANF_RESP "redeanfrage_antwort"

Definition at line 36 of file commands.hpp.

7.10.1.12 #define CMD_SET_PRAESENTATION "praesentation"

Definition at line 19 of file commands.hpp.

7.10.1.13 #define CMD_SET_SLIDE "set_slide"

Definition at line 18 of file commands.hpp.

7.10.1.14 #define CMD_STOP_PRAESENTATION "stop_praesentation"

Definition at line 20 of file commands.hpp.

7.10.1.15 #define CMD_UNKNOWN "unknown_command"

Definition at line 15 of file commands.hpp.

7.10.1.16 #define DATA_AUDIO "deliver_audio"

Definition at line 41 of file commands.hpp.

7.10.1.17 #define DATA_PRAESENTATION "deliver_praesentation"

Definition at line 21 of file commands.hpp.

7.11 Common/ClientServerShareLib/include/ConnectedClient.h File Reference

```
#include <QObject>
#include <QtNetwork/QTcpSocket>
#include <QtNetwork/QHostAddress>
#include <QByteArray>
```

Classes

- class Network::ConnectedClient

  Class for clients connected to the server.

Namespaces

- Network

7.12 Common/ClientServerShareLib/include/EMaudiorecorder.hpp File Reference

```
#include <bb/multimedia/AudioRecorder.hpp>
#include <bb/device/Led>
#include <bb/device/LedColor>
#include <QUrl>
#include <QTime>
#include <bb/system/InvokeDateTime>
#include <bb/multimedia/AudioChannelConfiguration.hpp>
#include <string.h>
#include <QObject>
#include <stdlib.h>
```

Classes

- class bb::EM2015::EMaudiorecorder

Namespaces

- bb
- bb::EM2015

7.13 Common/ClientServerShareLib/include/ExternalDisplay.hpp File Reference

```
#include <screen/screen.h>
#include <sys/platform.h>
#include <cstdint>
#include <iostream>
#include <QString>
#include <EGL/egl.h>
#include <GLES/gl.h>
#include <GLES/glext.h>
#include <bb/cascades/Image>
#include <bb/ImageData>
#include <bb/utility/ImageConverter>
#include <img/img.h>
```

Classes

- class ExternalDisplay

Typedefs

- typedef enum RENDERING_API RENDERING_API
- typedef enum VIEW_DISPLAY VIEW_DISPLAY
- typedef enum RESOLUTIONS_T RESOLUTIONS_T
- typedef enum DISPLAY_STATES_T DISPLAY_STATES_T

Enumerations

- enum RENDERING_API { GL_UNKNOWN = 0, GL_ES_1 = EGL_OPENGL_ES_BIT, GL_ES_2 = EGL_O↩ PENGL_ES2_BIT, VG = EGL_OPENVG_BIT }
- enum VIEW_DISPLAY { DISPLAY_UNKNOWN, DISPLAY_DEVICE, DISPLAY_HDMI }
- enum RESOLUTIONS_T { RES640x480, RES720x480, RES720x576, RES1024x768, RES1280x720, RES1280x1024, RES1600x1200, RES1920x1080 }
- enum DISPLAY_STATES_T { UNINITIALIZED, INITIALIZED }

7.13.1 Typedef Documentation

7.13.1.1 typedef enum DISPLAY_STATES_T DISPLAY_STATES_T

7.13.1.2 typedef enum RENDERING_API RENDERING_API

7.13.1.3 typedef enum RESOLUTIONS_T RESOLUTIONS_T

7.13.1.4 typedef enum VIEW_DISPLAY VIEW_DISPLAY

7.13.2 Enumeration Type Documentation

7.13.2.1 enum DISPLAY_STATES_T

Enumerator

UNINITIALIZED

INITIALIZED

Definition at line 32 of file ExternalDisplay.hpp.

7.13.2.2 enum RENDERING_API

Enumerator

GL_UNKNOWN
GL_ES_1
GL_ES_2
VG

Definition at line 29 of file ExternalDisplay.hpp.

7.13.2.3 enum RESOLUTIONS_T

Enumerator

RES640x480
RES720x480
RES720x576
RES1024x768
RES1280x720
RES1280x1024
RES1600x1200
RES1920x1080

Definition at line 31 of file ExternalDisplay.hpp.

7.13.2.4 enum VIEW_DISPLAY

Enumerator

DISPLAY_UNKNOWN
DISPLAY_DEVICE
DISPLAY_HDMI

Definition at line 30 of file ExternalDisplay.hpp.

7.14 Common/ClientServerShareLib/include/HDMI.hpp File Reference

```
#include "ExternalDisplay.hpp"
#include <QUrl>
#include <bb/cascades/Image>
#include <bb/ImageData>
#include <bb/utility/ImageConverter>
#include <img/img.h>
```

Classes

- class bb::EM2015::HDMI

Namespaces

- bb
- bb::EM2015

7.15 Common/ClientServerShareLib/include/Message.hpp File Reference

```
#include <QDateTime>
#include <QObject>
#include <QRegExp>
#include "XMLMessageWriter.hpp"
```

Classes

- class Message

  Implementation of a Message.

Macros

- #define MESSAGE_DATETIME_FORMAT "dd.MM.yyyy hh:mm:ss.zzz"

7.15.1 Macro Definition Documentation

7.15.1.1 #define MESSAGE_DATETIME_FORMAT "dd.MM.yyyy hh:mm:ss.zzz"

Definition at line 20 of file Message.hpp.

7.16 Common/ClientServerShareLib/include/MessageAuthenticator.h File Reference

```
#include <QObject>
#include <QCryptographicHash>
```

Classes

- class MessageAuthenticator

  Provides a transparent interface to authenticate messages via HMAC given a key.

7.17 Common/ClientServerShareLib/include/Praesentation.hpp File Reference

```
#include <QtCore>
#include <QObject>
#include <bb/cascades/Image>
#include <bb/utility/ImageConverter>
#include <bb/ImageData>
#include "Message.hpp"
#include "commands.hpp"
```

Classes

- class Praesentation

  Class to hold information about the presentation.

7.18 Common/ClientServerShareLib/include/Redeanfrage.hpp File Reference

```
#include <QObject>
#include "Message.hpp"
#include "commands.hpp"
```

Classes

- class Redeanfrage

  Class containing one talk request and its state.

7.19 Common/ClientServerShareLib/include/RedeanfrageQueue.hpp File Reference

```
#include <QObject>
#include <QtCore>
#include "Redeanfrage.hpp"
```

Classes

- class RedeanfrageQueue

  Thread safe implementation of a queue containing talk requests.

7.20 Common/ClientServerShareLib/include/ServerSocket.h File Reference

```
#include <QObject>
#include <QtNetwork/QTcpServer>
#include <QtNetwork/QTcpSocket>
#include <QString>
#include <QByteArray>
#include <QList>
#include "ConnectedClient.h"
```

Classes

- class Network::ServerSocket

  ServerSocket class.

Namespaces

- Network

7.21 Common/ClientServerShareLib/include/XMLMessageParser.hpp File Reference

```
#include <QObject>
#include <QtCore>
#include <QDateTime>
#include "Message.hpp"
```

Classes

- class XMLMessageParser

  Parses Messages from XML to Message Object. Capable of separating between data an command/seperation by parent class. Throws signal if message was parsed.

7.22 Common/ClientServerShareLib/include/XMLMessageWriter.hpp File Reference

```
#include <QObject>
#include <QtCore>
#include "Message.hpp"
```

Classes

- class XMLMessageWriter

  Serializes Message object to XML. Capable of separating between data an command/seperation by parent class. Throws signal if message was written.

7.23 Common/ClientServerShareLib/src/AudioRecorder/EMaudiorecorder.cpp File Reference

```
#include "include/EMaudiorecorder.hpp"
```

Namespaces

- bb
- bb::EM2015

7.24 Common/ClientServerShareLib/src/Camera/CameraController.cpp File Reference

```
#include "include/CameraController.hpp"
```

7.25 Common/ClientServerShareLib/src/Camera/CameraProcessor.cpp File Reference

```
#include "include/CameraProcessor.hpp"
```

7.26 Common/ClientServerShareLib/src/Client/Client.cpp File Reference

```
#include "include/Client.hpp"
```

7.27 Common/ClientServerShareLib/src/HDMI/ExternalDisplay.cpp File Reference

```
#include "include/ExternalDisplay.hpp"
```

7.28 Common/ClientServerShareLib/src/HDMI/HDMI.cpp File Reference

```
#include "include/HDMI.hpp"
```

Namespaces

- bb
- bb::EM2015

7.29 Common/ClientServerShareLib/src/Message/Authentication/MessageAuthenticator.cpp File Reference

```
#include "include/MessageAuthenticator.h"
```

7.30 Common/ClientServerShareLib/src/Message/Message.cpp File Reference

```
#include "include/Message.hpp"
```

7.31 Common/ClientServerShareLib/src/Message/XML/XMLMessageParser.cpp File Reference

```
#include "include/XMLMessageParser.hpp"
```

7.32 Common/ClientServerShareLib/src/Message/XML/XMLMessageWriter.cpp File Reference

```
#include "include/XMLMessageWriter.hpp"
```

7.33 Common/ClientServerShareLib/src/Network/ClientSocket.cpp File Reference

```
#include "include/ClientSocket.h"
#include <QDebug>
#include <QHostAddress>
#include <QIODevice>
```

Namespaces

- Network

7.34 Common/ClientServerShareLib/src/Network/ConnectedClient.cpp File Reference

```
#include "include/ConnectedClient.h"
#include <QDebug>
#include <QHostAddress>
#include <QIODevice>
```

Namespaces

- Network

7.35 Common/ClientServerShareLib/src/Network/ServerSocket.cpp File Reference

```
#include "include/ServerSocket.h"
#include <QDebug>
#include <QtNetwork/QTcpSocket>
#include <QtNetwork/QHostAddress>
#include <QtNetwork/QNetworkInterface>
#include <QThread>
```

Namespaces

- Network

7.36 Common/ClientServerShareLib/src/Praesentation/Praesentation.cpp File Reference

```
#include "include/Praesentation.hpp"
```

7.37 Common/ClientServerShareLib/src/Redeanfrage/Redeanfrage.cpp File Reference

```
#include "include/Redeanfrage.hpp"
```

7.38 Common/ClientServerShareLib/src/Redeanfrage/RedeanfrageQueue.cpp File Reference

```
#include "include/RedeanfrageQueue.hpp"
```

7.39 ServerAppl/src/backend/ByteStreamVerifier.cpp File Reference

```
#include <src/backend/ByteStreamVerifier.h>
#include <src/backend/Logger.h>
```

Namespaces

- ServerAppl

7.40 ServerAppl/src/backend/ByteStreamVerifier.h File Reference

```
#include <QMap>
#include <QObject>
#include <MessageAuthenticator.h>
```

Classes

- class ServerAppl::ByteStreamVerifier

  An object of this class can verify a QByteStream that has a hash at its and.

Namespaces

- ServerAppl

7.41 ServerAppl/src/backend/Listener.cpp File Reference

```
#include <Qt>
#include <src/backend/Listener.h>
#include <commands.hpp>
#include <src/backend/Logger.h>
#include <src/backend/UnspecifiedClient.h>
```

Namespaces

- ServerAppl

7.42 ServerAppl/src/backend/Listener.h File Reference

```
#include <QObject>
#include <Message.hpp>
#include <src/backend/UnspecifiedClient.h>
#include <src/backend/MessageHandlerInterface.h>
```

Classes

- class ServerAppl::Listener

  Objects of this class represent a listener-client.

Namespaces

- ServerAppl

7.43 ServerAppl/src/backend/Logger.cpp File Reference

```
#include <QDate>
#include <QDir>
#include <QTime>
#include <src/backend/Logger.h>
```

Namespaces

- ServerAppl

Variables

- ServerAppl::Logger serverLogObject

7.43.1 Variable Documentation

7.43.1.1 ServerAppl::Logger serverLogObject

Definition at line 14 of file Logger.cpp.

7.44 ServerAppl/src/backend/Logger.h File Reference

```
#include <QDebug>
#include <QFile>
#include <QString>
#include <QTextStream>
```

Classes

- class ServerAppl::Logger

Namespaces

- ServerAppl

Macros

- #define IS_DEBUG_VERSION
- #define WRITE_LOG(msg) serverLogObject.writeLogEntry(QString(msg));
- #define WRITE_DEBUG(msg) serverLogObject.writeDebugLogEntry(__FILE__, __LINE__, QString(msg)); qDebug() << __FILE__ << ":" << __LINE__ << " --> " << msg;

Variables

- ServerAppl::Logger serverLogObject

7.44.1 Macro Definition Documentation

7.44.1.1 #define IS_DEBUG_VERSION

Definition at line 16 of file Logger.h.

7.44.1.2 #define WRITE_DEBUG(*msg*) serverLogObject.writeDebugLogEntry(__FILE__, __LINE__, QString(msg)); qDebug() << __FILE__ << ":" << __LINE__ << " --> " << msg;

Definition at line 49 of file Logger.h.

7.44.1.3 #define WRITE_LOG(*msg*) serverLogObject.writeLogEntry(QString(msg));

Definition at line 46 of file Logger.h.

7.44.2 Variable Documentation

7.44.2.1 ServerAppl::Logger serverLogObject

Definition at line 14 of file Logger.cpp.

7.45 ServerAppl/src/backend/Master.cpp File Reference

```
#include <commands.hpp>
#include <src/backend/Logger.h>
#include <src/backend/Master.h>
```

Namespaces

- ServerAppl

7.46 ServerAppl/src/backend/Master.h File Reference

```
#include <QByteArray>
#include <QObject>
#include <QTimer>
#include <Praesentation.hpp>
#include <MessageAuthenticator.h>
#include <src/backend/UnspecifiedClient.h>
#include <src/backend/MessageHandlerInterface.h>
```

Classes

- struct NONCE
- class ServerAppl::Master

  Objects of this class represent a master-client.

Namespaces

- ServerAppl

Enumerations

- enum MasterAuthenticationState { PHASE_1, PHASE_2, PHASE_3, ACCEPTED, REJECTED }
- enum MasterAuthenticationEvent { ReceivedPhase1Message, TransmittedPhase2Message, ReceivedPhase3Message, ReceivedAcknowledgeMessage, ReceivedCorruptAuthenticationMessage }

7.46.1 Enumeration Type Documentation

7.46.1.1 enum MasterAuthenticationEvent

Enumerator

ReceivedPhase1Message

TransmittedPhase2Message

ReceivedPhase3Message

ReceivedAcknowledgeMessage

ReceivedCorruptAuthenticationMessage

Definition at line 35 of file Master.h.

7.46.1.2 enum MasterAuthenticationState

Enumerator

PHASE_1

PHASE_2

PHASE_3

ACCEPTED

REJECTED

Definition at line 26 of file Master.h.

7.47 ServerAppl/src/backend/MessageHandlerInterface.h File Reference

```
#include <QString>
#include <Message.hpp>
```

Classes

- class ServerAppl::MessageHandlerInterface

Namespaces

- ServerAppl

Macros

- #define IS_COMMAND(cmdObj, cmdStr) ((cmdStr) == (cmdObj))
- #define HANDLER_OBJ(obj) ((MessageHandlerInterface *)(obj))
- #define HANDLER_FUNC(func) (static_cast<handleReceivedMessageFunction>(&func))

7.47.1 Macro Definition Documentation

7.47.1.1 #define HANDLER_FUNC(*func*) (static_cast<handleReceivedMessageFunction>(&func))

Definition at line 17 of file MessageHandlerInterface.h.

7.47.1.2 #define HANDLER_OBJ(*obj*) ((MessageHandlerInterface *)(obj))

Definition at line 16 of file MessageHandlerInterface.h.

7.47.1.3 #define IS_COMMAND(*cmdObj, cmdStr*) ((cmdStr) == (cmdObj))

Definition at line 15 of file MessageHandlerInterface.h.

7.48 ServerAppl/src/backend/MessageRouter.cpp File Reference

```
#include <src/backend/Logger.h>
#include <src/backend/MessageRouter.h>
```

Namespaces

- ServerAppl

7.49 ServerAppl/src/backend/MessageRouter.h File Reference

```
#include <QMap>
#include <QObject>
#include <QString>
#include <Message.hpp>
#include <src/backend/MessageHandlerInterface.h>
```

Classes

- struct messageHandler
- class ServerAppl::MessageRouter

  This class receives Message-objects and delivers them to target-objects.

Namespaces

- ServerAppl

Typedefs

- typedef Message *(ServerAppl::MessageHandlerInterface::* handleReceivedMessageFunction) (QString commandName, Message *message)

7.49.1 Typedef Documentation

7.49.1.1 typedef Message*(ServerAppl::MessageHandlerInterface::* handleReceivedMessageFunction) (QString commandName, Message *message)

Definition at line 19 of file MessageRouter.h.

7.50 ServerAppl/src/backend/Server.cpp File Reference

```
#include <src/backend/Server.h>
#include <commands.hpp>
#include <src/backend/Listener.h>
#include <src/backend/Master.h>
#include <src/backend/Logger.h>
```

Namespaces

- ServerAppl

7.51 ServerAppl/src/backend/Server.h File Reference

```
#include <QHostAddress>
#include <QList>
#include <QMap>
#include <QObject>
#include <Message.hpp>
#include <Praesentation.hpp>
#include <ServerSocket.h>
#include <XMLMessageParser.hpp>
#include <XMLMessageWriter.hpp>
#include <src/backend/ByteStreamVerifier.h>
#include <src/backend/Listener.h>
#include <src/backend/Master.h>
#include <src/backend/MessageHandlerInterface.h>
#include <src/backend/MessageRouter.h>
```

Classes

- class ServerAppl::Server

  An object of this class is the central component of the server-application.

Namespaces

- ServerAppl

7.52 ServerAppl/src/backend/UnspecifiedClient.cpp File Reference

```
#include <src/backend/UnspecifiedClient.h>
#include <commands.hpp>
#include <Message.hpp>
#include <src/backend/Logger.h>
#include <src/backend/Master.h>
#include <src/backend/Server.h>
```

Namespaces

- ServerAppl

7.53 ServerAppl/src/backend/UnspecifiedClient.h File Reference

```
#include <QObject>
#include <QString>
#include <QTime>
#include <Message.hpp>
#include <src/backend/MessageHandlerInterface.h>
```

Classes

- class ServerAppl::UnspecifiedClient

  Objects of this class represent a client the has not shown if its a master-client or listener-client.

Namespaces

- ServerAppl

Enumerations

- enum ClientType { ClientType_Unspecified, ClientType_Listener, ClientType_Master }

7.53.1 Enumeration Type Documentation

7.53.1.1 enum ClientType

Enumerator

ClientType_Unspecified

ClientType_Listener

ClientType_Master

Definition at line 19 of file UnspecifiedClient.h.